

NASA Contractor Report 166070

FAULT-TOLERANT SOFTWARE FOR THE FTMP

Herbert Hecht and Myron Hecht

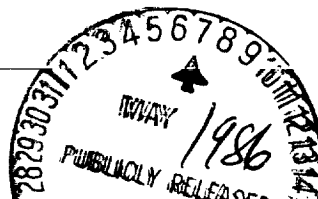
Prepared for

THE CHARLES STARK DRAPER LABORATORY, INC.
Cambridge, Massachusetts

By

SoHar Incorporated
Los Angeles, California

Final Engineering Report
Subcontract 564
Prime Contract NAS1-15336
March 1984



{NASA-CR-166070} FAULT-TOLERANT SOFTWARE
FOR THE FTMP Final Report {SoHar, Inc.}
82 p HC A05/MF A01 CSCL 09B

N86-24276

Unclas

G3/62 06031

Review for general release March, 1986

COPY CONTROL NO.

NASA

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

COPY CONTROL NUMBER. THE RECIPIENT IS

TABLE OF CONTENTS

1. Introduction.....	1
1.1. Recovery Blocks.....	3
1.2. Functional Description of the Dispatcher.....	6
1.3. Coverage of the Primary Routine Failures.....	17
1.4. Software Errors Not Covered By Dispatcher Acceptance Tests.....	24
2. Functional Acceptance Tests.....	25
2.1. Dispatcher Acceptance Tests.....	27
2.2. Interval Timer Acceptance Test.....	37
2.3. Input/Output Acceptance Tests.....	40
2.4. Applications Routines.....	43
3. Structural Acceptance Tests.....	44
3.1. Errors In SLIP and R.DONE.....	45
3.2. STUCK IN R4 Acceptance Test.....	48
3.3. KICK Acceptance Test and Modifications to KICK Procedures.....	51
3.4. R4 Responsible Acceptance Test.....	53
3.5. Uninterruptible Code Acceptance Test.....	56
3.6. Retirement Acceptance Test.....	62
4. Alternate Dispatcher.....	65
4.1. Alternate Dispatcher Requirements.....	65
4.2. Description of the Alternate Dispatcher.....	68
References.....	70
Appendix A: New Variables Required.....	71
Appendix B: Uninterruptible ASM routines.....	73

List of Figures

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1.1	Implementation of the Recovery Block for the FTMP	4
1.2.	R4 Frame Initiation -- Interrupting R3 and R1	8
1.3.	R4 Frame Termination -- Resume R3	10
1.4.	R4 Task Invocation Procedure Control Flow	12
1.5.	R3 and R1 Task Invocation Procedure Control Flow	13
1.6.	Pend and Activate Handling	15
1.7.	Interrupt Handling	16
1.8.	Top Level Fault Tree for the FTMP Dispatcher	18
1.9.	Initialization Faults	20
1.10.	Execution Order Failures	21
1.11.	Timing Failures	22
1.12.	Recovery Block Faults	23
2.1.	Sequence of Timer Interrupts and Dispatcher Acceptance Tests	26
2.2.	Fault Tolerant Provisions for the FTMP Dispatcher	30
2.3.	Dispatcher Critical Word Acceptance Test Module	31
2.4.	Frame Count Acceptance Test Module	33
2.5.	Frame Counters	35
2.6.	Critical Word Reset Acceptance Test Module	36
2.7.	Interval Timer Acceptance Test	39

PRECEDING PAGE BLANK NOT FILMED

List of Figures (continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.1.	SLIP Acceptance Test	47
3.2.	R.DONE Acceptance Test	47
3.3.	STUCK.IN.R4 Acceptance Test	50
3.4.	KICK Acceptance Test	50
3.5.	R4.RESPONSIBLE Acceptance Test	55
3.6.	Uninterruptible Code Acceptance Test: Triad Working	59
3.7.	Uninterruptible Code Acceptance Test: Triad Idling	61
3.8.	Retirement Acceptance Test	64
4.1.	Alternate Dispatcher	69

SECTION 1 - INTRODUCTION

This is the Final Engineering Report prepared for The Charles Stark Draper Laboratory, Inc. under Subcontract 564 (Prime Contract NAS1-15336), covering technical assistance in fault tolerant software development for the Fault Tolerant Multiprocessor (FTMP). This report satisfies item 4.5 of the subcontract.

The FTMP is a highly reliable computer intended to service reliability-critical applications in scheduled aircraft service. Work on the architecture to provide the required hardware fault-tolerance has been in progress since the mid-sixties, and has evolved into a well-understood highly redundant system described in ref. 1. Although this design effectively addresses the detection, masking, and elimination of hardware faults, it can not circumvent failures due to software faults.

The work reported on here provides protection against software failures in the dispatcher of the FTMP, a particularly critical portion of the system software. Faults in other system modules and in application programs can be handled by similar techniques but coverage for these was not provided in this effort. Goals of the work reported on here are (1) to develop provisions in the software design that will detect and mitigate software failures in the dispatcher portion of the system executive and (2) to propose the implementation of specific software reliability measures in other parts of the system. In proceeding toward these goals, the following constraints were observed:

- the coverage of the dispatcher was to be complete; no potential failure modes were to be overlooked due to difficulty of implementation, and

- the additional software required for implementation of fault tolerance was to be simple, to minimally affect the design and the operation of the current system, and to minimize the introduction of new variables.

All of these requirements have been met by the design described in later sections of this report, and a basis has therefore been provided for augmenting the hardware fault tolerance provisions of the FTMP with equally effective measures for software fault tolerance.

Beyond the specific support to the FTMP project, the work reported on here represents a considerable advance in the practical application of the recovery block methodology for fault tolerant software design. The operations carried out by the dispatcher are primarily in the area of logic and sequencing, and error detection techniques for such operations are more difficult than for programs dealing with flight data for which reasonableness tests based on physical constraints can be devised. The acceptance tests for dispatcher functions had to be based on logic that was independent of the logic used in the primary program.

In pursuing the goal of independent acceptance tests, it was found convenient to divide these into two classes: functional acceptance tests which determine compliance with program requirements, and structural acceptance tests which

determine adherence to a predefined logic flow. These tests, as devised for the FTMP, are described in Sections 2 and 3, respectively, and they represent an implementation for a particularly challenging environment. When these tests are not satisfied, the program either retries or invokes an alternate dispatcher directly. The design for the latter, described in Section 4, is entirely independent of that of the primary dispatcher. Both the hardware and software structure is less flexible (and therefore less efficient) but correspondingly more rugged. Because most software failures are not permanent, the alternate dispatcher will attempt reversion to the primary one at discrete intervals during routine operation. In this way, the reduced efficiency of the alternate will in most cases have very little effect on the operation of the flight programs.

Before leaving this part of the Introduction, the authors wish to express their thanks for the cooperation received in this effort from personnel of The Charles Stark Draper Laboratory and of the NASA Langley Research Center. Dr. Albert L. Hopkins gave whole-hearted support to this work and made many helpful suggestions. Drs. Basil T. Smith and Jay Lala facilitated the design of the fault tolerant software by making the design and data for the primary dispatcher available and patiently explaining details when this was required. To Mr. Billy L. Dove and Mr. Nicholas D. Murray our thanks for the support of this work and for permitting us to participate in an important area of fault tolerant computing.

1.1. RECOVERY BLOCKS

The inability to perform conclusive reliability evaluations on software motivates the development of fault-tolerance techniques. Two techniques for achieving fault-tolerance have been discussed in the recent literature: the recovery block and N-version programming. N-version programming involves a number (at least two) of independently coded programs for a given function that are run simultaneously (or nearly so) on loosely coupled computers. The results are compared, and in case of a disagreement, a preferred result is identified by a majority vote (for three or more versions) or a predetermined strategy.

The second technique, and the one used in the FTMP, is the recovery block (refs. 2 and 3). The simplest structure for the recovery block is

Ensure T

By P

Else by Q

Else Error

where T is an acceptance test condition, i.e. a condition which is expected to be met by successful execution of a primary routine, P, or the alternate routine Q. The internal control structure of the recovery block will transfer to Q when the test conditions are not met by executing P.

Implementation of recovery blocks for the FTMP is shown in fig. 1. The primary routine is executed, and if the acceptance test conditions are not met, the alternate routine is invoked. The number of passes through the alternate routine is counted, and after a predetermined limit (dependent on the capabilities of the primary and alternate programs, execution time, and other factors), a transition is made back to the primary routine.

The key element of the recovery block approach is the acceptance test. There are two levels on which acceptance tests can be performed. The first is the functional level, i.e. that which tests that the outputs of the program are consistent with the functional requirements. The second is the structural level, which tests sections of code to ensure that key variables and functions have been properly executed. Functional level tests are appropriate for software that has been in use a long time because they are simpler and it avoids unnecessary transfers. However, for programs under development, the addition of structural tests provide the following benefits:

1. Unexpected behavior of the primary systems will be noted even in cases where only a mild degradation is encountered. This aids in program evaluation.

ORIGINAL PAGE 19
OF POOR QUALITY

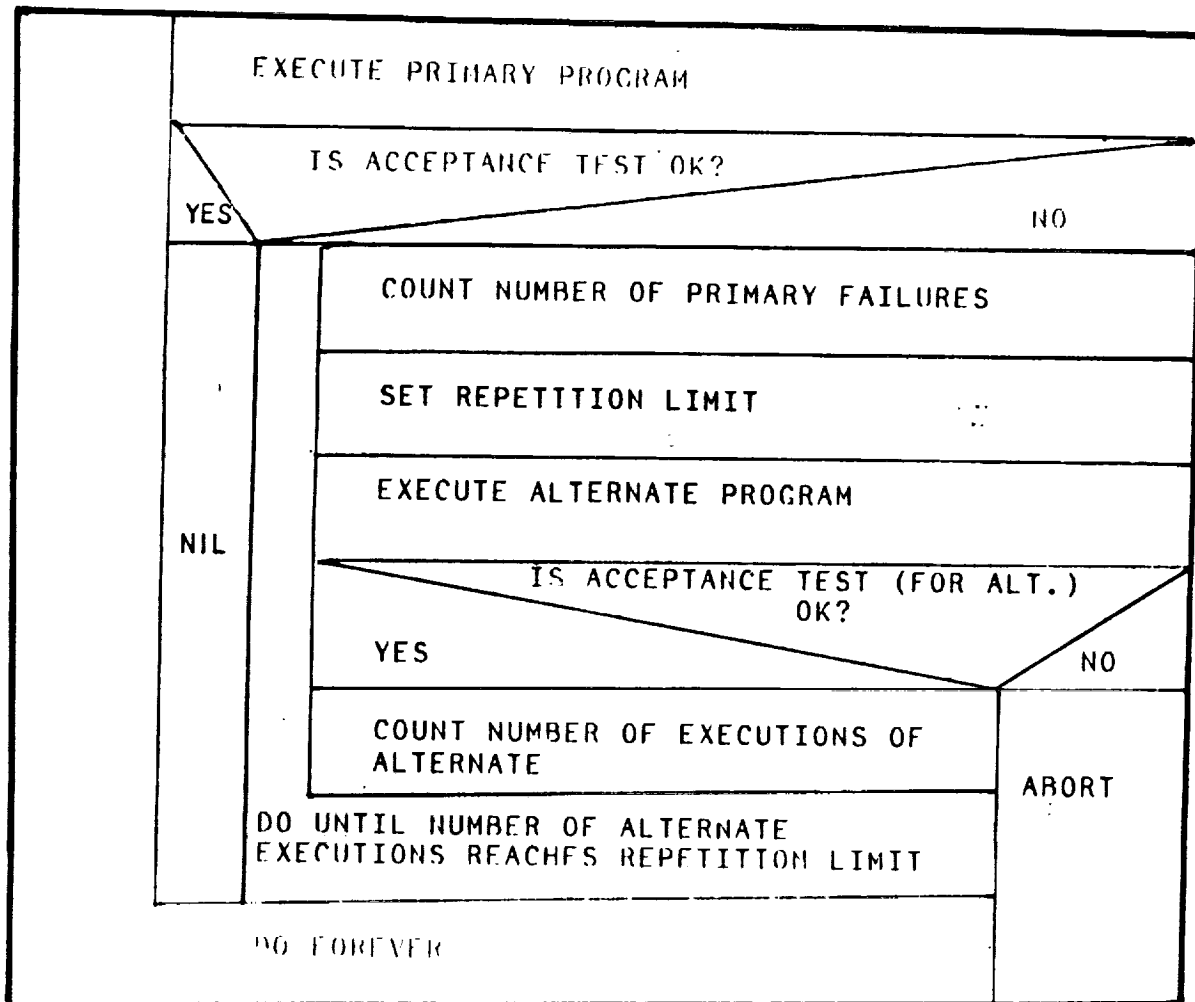


FIGURE 1.1 Implementation of the Recovery Block for the FTMP

2. Switching to the alternate program is exercised more often under realistic (unplanned) conditions. Providing realistic testing of the fault tolerance mechanism is a difficult undertaking.
3. As a program matures, it is usually easier to relax acceptance conditions than to make them more restrictive.

1.2. FUNCTIONAL DESCRIPTION OF THE DISPATCHER

The FTMP task dispatching function selects an applications routine for an available processor, relinquishes control of the processor for a set period to enable task execution, and then returns to select the next task or pass control to a lower priority executive process. By virtue of its multiprocessor arrangement, the execution order of tasks is not fixed, and thus, the dispatcher must perform the following functions: (1) determine how frequently a task should be run, (2) determine whether data and predecessor tasks have been completed, (3) invoke the task and enable interrupts for higher priority items or overtime, and (4) maintain records of functions which have been executed along with other housekeeping tasks. The centrality of this function to the FTMP operation made it a prime candidate for the implementation of software fault-tolerance measures.

Because the scope of the implementation of fault-tolerance was limited to the dispatcher and associated routines, the design of acceptance tests and of the alternate dispatcher was based on a portion of the entire system executive. This section presents the functional specifications of relevant portions of the FTMP operating system upon which this report rests.

The dispatcher is divided into two major routines: the R4 rate group dispatcher, designated as R4.DISPATCHER, and the R3 and R1 rate groups dispatcher, designated as R3.R1.DISPATCHER. The R4 dispatcher performs five major functions:

1. Initialization during system start up
2. R4 frame initiation
3. Reading error latches and performing I/O to the 1553 bus
4. Task selection and execution
5. Retirement

The R3.R1.DISPATCHER relies on the R4 dispatcher for the above functions, and thus, it only performs task selection and execution for the appropriate rate groups. Section 1.2.1 contains a more detailed description of the means by which the dispatcher routine performs these functions.

A crucial requirement of the dispatcher is that it perform its functions within strict timing limits. Thus, a number of routines external to the dispatcher are necessary to maintain the proper time references and to perform interruptions at appropriate intervals. These routines are discussed in section 1.2.2.

1.2.1. Dispatcher functions

Initialization

Initialization consists of restoring portions of the system memory to default values and zeroing out other portions. Initialization is entered upon a restart flag being set to TRUE, and results in the following:

- The triad to start initialization becomes R4.RESPONSIBLE
- The system timer is initialized
- Triad busy states are set to free
- Triad status words are set to enable execution of all rate groups
- Reconfiguration commands and states are set to 0
- The configuration controller is initialized
- Unlock and IPC interrupt commands are set to 0
- Error latches are cleared
- The rate group control blocks for all dispatchers are initialized
- Lower rate groups are set to be executed in later frames

R4 frame initiation

A new R4 frame is started every 40 msec. At the start of a new frame, the R4 dispatcher takes the following actions:

- lower rate groups are interrupted and their timers are stopped
- the triad states are updated
- task pointers are set to begin at the top of the R4 list
- lower rate groups are pended for execution at the appropriated frames
- I/O for the appropriate rate groups is performed
- error latches are read and cleared
- reconfiguration occurs if any commands are pending

The start of a new R4 frame takes precedence over all other actions. FTMP senses the start of a new time frame when a triad responsible for the frame restart, designated as R4.RESPONSIBLE, responds to a timer interrupt. Figure 1.2, adopted from a CSDL briefing chart, shows the sequence of events after the R4 interrupt. The R4 responsible triad, shown in the center, is the only one whose timer is set with the new frame time. After it performs the initialization, it sends an Interprocessor (IPC) interrupt to a second triad, which is shown as being in the idle mode. This second triad then starts up the R4 dispatcher for itself, and sends an IPC interrupt to the third triad shown executing the R1 rate group. The third triad will interrupt execution of the current process, freeze the timers and other relevant control variables, and restart the R4 dispatcher.

Task selection and execution

After completion of the R4 frame initialization tasks, the R4 dispatcher enters a task selection loop using an internal procedure SELECT.TASK. All tasks to be executed in the R4 rate group are contained in a list. Elements of this list, denoted as task control blocks (TCBs), contain information on preceeding and succeeding tasks, time limit, and most recent execution. Additional pointer variables lead to data buffers for I/O for each applications task and to

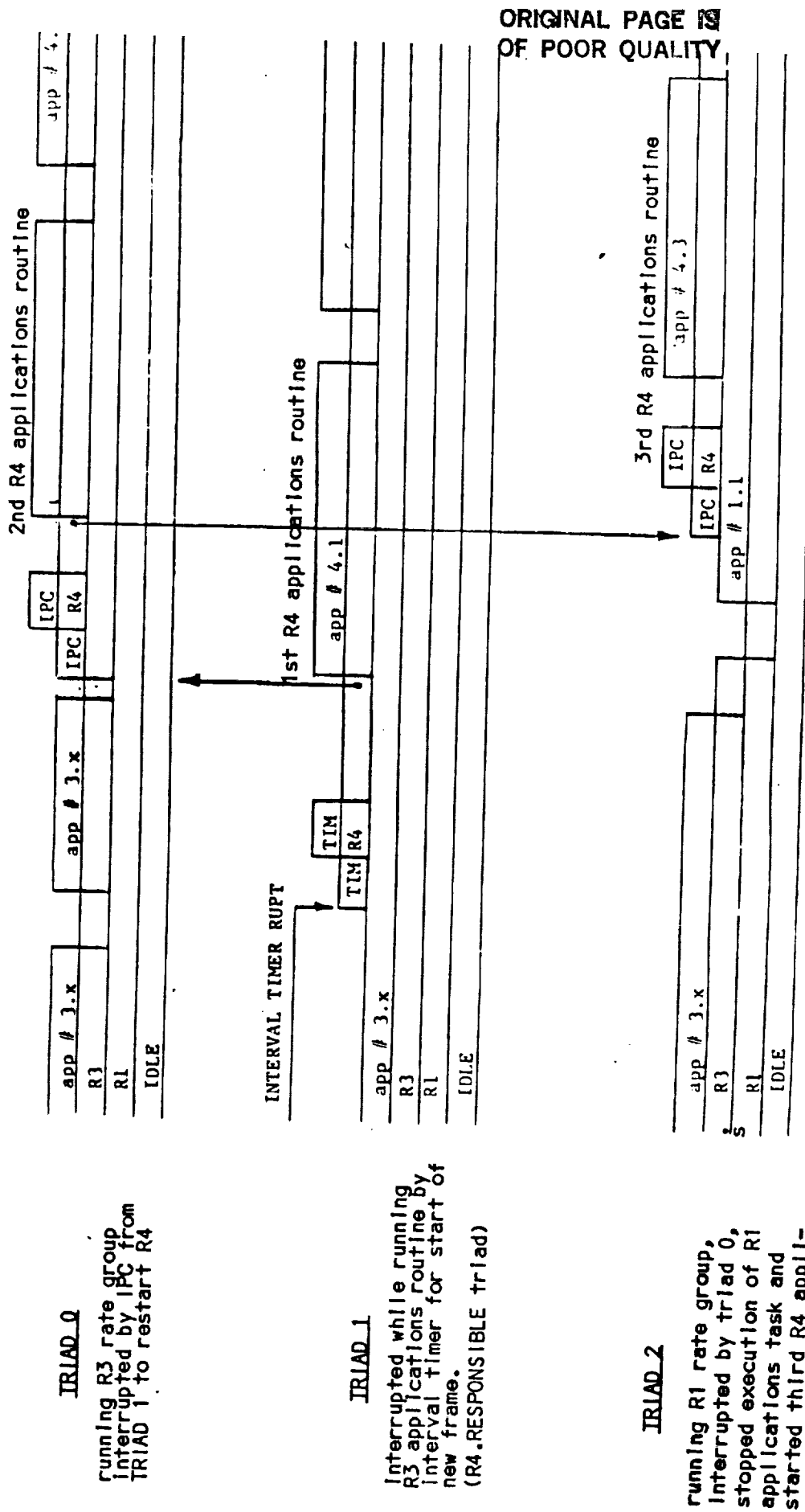


FIGURE 1.2. R4 Frame Initiation -- Interrupting R3 and R1

constraints, i.e. tasks which must be executed prior to the current one.

Task execution occurs by means of an internal procedure EXECUTE which reads data areas from the buffer locations designated by the TCBs from main memory into the triad cache memory, starts the R4 timer, and passes control to the task by means of an ASM procedure ACTIVATE. If the task has not run over its time limit, control will be passed back to EXECUTE upon its completion which updates the frame count in the TCB and sets an appropriate bit in the constraint word to indicate task completion.

Applications routine selection and execution continues in this manner until SELECT.TASK finds a null value for a succeeding task pointer. At this point the triad notes that the task list is done in system memory and sets itself as responsible for starting the next R4 frame, updates the R4 task list in main memory, and restarts timers and lower rate groups which were interrupted at the beginning of the current frame. Figure 1.3, also adopted from a CSDL briefing, shows the return of the system to lower rate group tasks. If no tasks were interrupted, the triad goes to an idle process until the next interrupt.

Retirement

The final function of the R4 dispatcher is to recognize the retire command generated by the configuration controller. Retirement means that a triad has sustained a permanent failure and will no longer be able to execute tasks. Retirement involves setting the following variables:

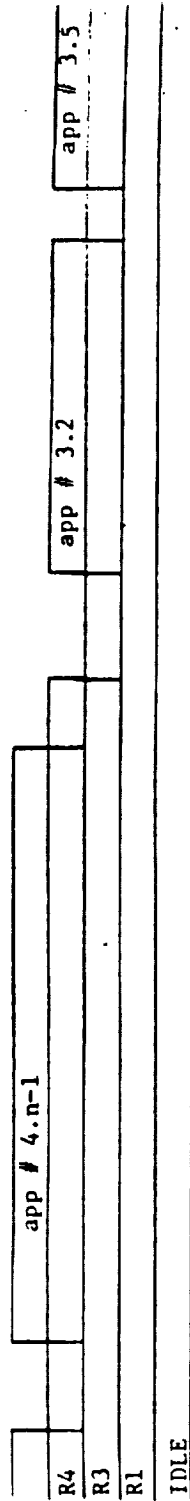
- triad status bit to prohibit execution of the R4 rate group
- setting the triad busy word to indicate this triad is not working
- decrementing the R4 triad counter
- initiating the idle process

If the retiring triad is also R4.RESPONSIBLE, it must restart the R4 dispatcher in another triad in order to provide another triad with the responsibility of setting the time to and starting the next frame. The means of performing this change is a higher priority IPC interrupt, which will cause the receiving triad to halt execution of its previous task until the interrupt is handled.

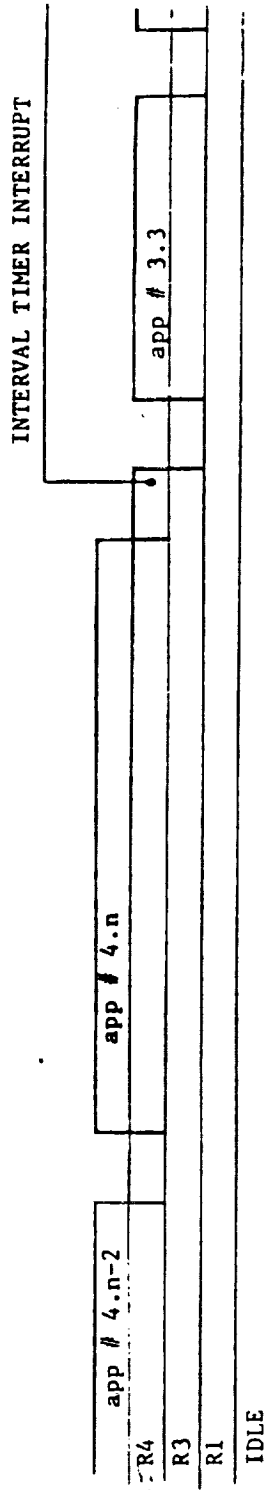
Lower rate group dispatchers

The R3 and R1 rate groups have task selection and execution functions similar to the R4 rate group, but are not responsible for restart, I/O, retirement, or initialization. An additional function of the lower rate groups is the detection of a need to continue execution of their applications routines beyond the time allowed in the current frame. Should the task list complete markers for the lower rate groups not be set to TRUE, a variable designated as SLIP will be decremented. The decrementing of SLIP will, when added to the frame count, have the effect of delaying the restart of the R3 rate group by one frame, and the R1 rate group by two frames.

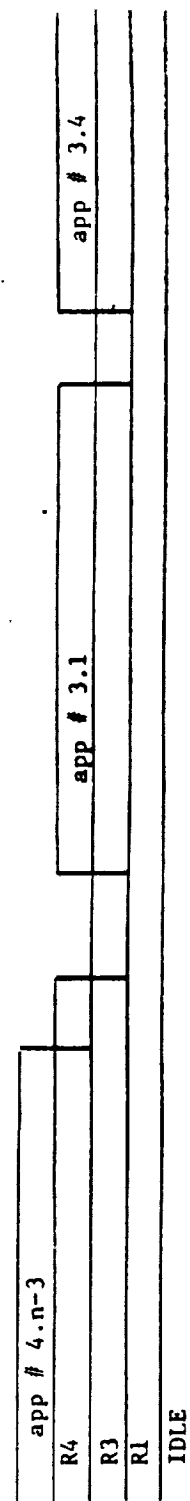
1.2.2. Timing routines associated with the Dispatchers



TRIAD 0
completed its
R4 task, could
not find any
others, and
resumed R3 execu-
tion



TRIAD 1
selected the last
R4 task and
became
R4.RESPONSIBLE



TRIAD 2
completed its
R4 task, could
not find any
others, and
resumed R3 execu-
tion

FIGURE 1.3. R4 Frame Termination -- Resume R3

Figures 1.4 and 1.5 show the sequence of execution of various procedures associated with the R4 and lower rate group dispatchers respectively. Those associated with timing are underlined. As is evident from both figures, triads act on the basis of time-generated interrupts. The time to the interrupts is placed in a register denoted as the INTERVAL TIMER and decremented at 250 microsecond intervals until it reaches zero. At this point a timer interrupt is generated.

At the beginning of a new frame, the R4 rate group dispatcher will save the times left for interrupted lower rate group executions by means of the HOLD.R3.R1.TIMERS routine. As shown in figure 1.4, the R4 dispatcher causes the activation of the lower rate groups by pointing to the addresses of their process state descriptors (figure 1.6) at the appropriate frames. After completing frame initialization, the dispatchers invoke applications routines by means of the SELECT and EXECUTE procedures described above. Prior to passing control to the applications routine, the dispatcher starts a timer which will interrupt execution should the applications routine run over the time limit written in its task control block. If an interrupt occurs, a routine designated as the TIMER.INTERRUPT.HANDLER is executed. This procedure determines whether a task time-out or a new frame interrupt occurred (if the triad is R4 responsible), and sets up the R4 dispatcher to be restarted in the latter case. In the absence of an interrupt, the dispatcher repeats the process with subsequent applications routines until the list is complete.

Upon completion of the R4 iteration, the timers of the lower rate groups are restarted by the RELEASE.R3.R1.TIMERS procedure, and control is passed to the R3 dispatcher by means of a RESUME statement and the PEND procedure executed at the beginning of the frame. The R1 timer is saved by means of the HOLD.R1.TIMER routine, and the R3 rate group applications routines are selected, executed, and interrupted (if necessary) in the same manner as were the R4 applications tasks. The R3.TIMER routines are somewhat more complicated because the times they place in the interval timers must also observe the total frame time limit, i.e. if the time allowed for an applications routine is greater than the time remaining in the frame, the interval timer is set with the time remaining in the frame.

Upon completion of the R3 dispatcher in a frame where R1 is to be run, the R1 rate group timer is released and the R1 dispatcher is invoked as described above. Execution of the R1 rate group dispatcher and applications routines is identical to the R3 execution with the single exception of R1 timer routines rather than R3 timer routines used for starting and stopping the interval timer. Because no rate groups are executed behind R1, there is no need to hold or release other timers.

1.2.3. Execution Order

Each resident process within the triad has a process state descriptor (PSD) resident in the cache memory which contains a pointer to the location of the PSD for a succeeding task. In this manner, a number of tasks can be arranged to

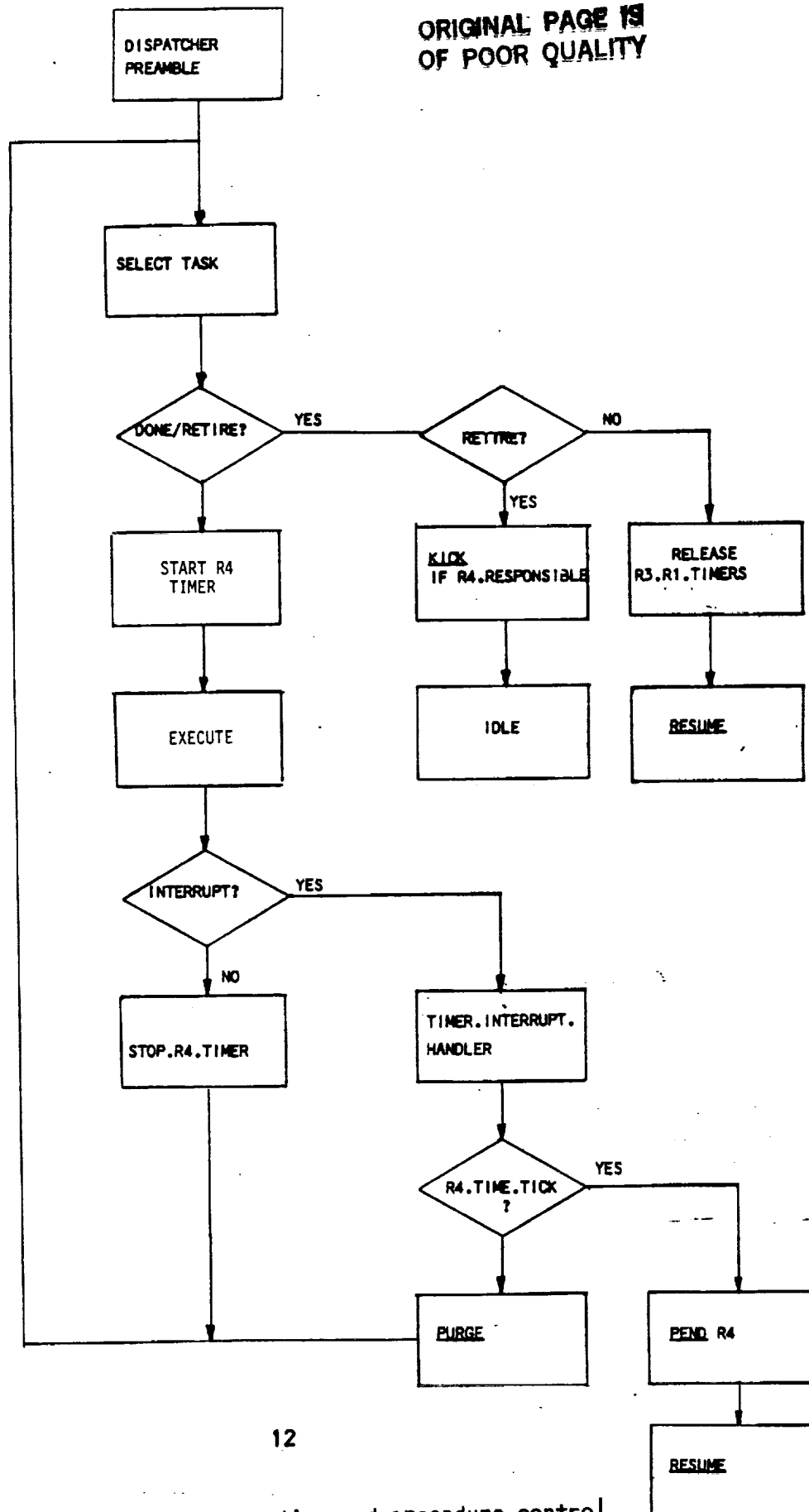


Figure 1.4. R4 task invocation and procedure control

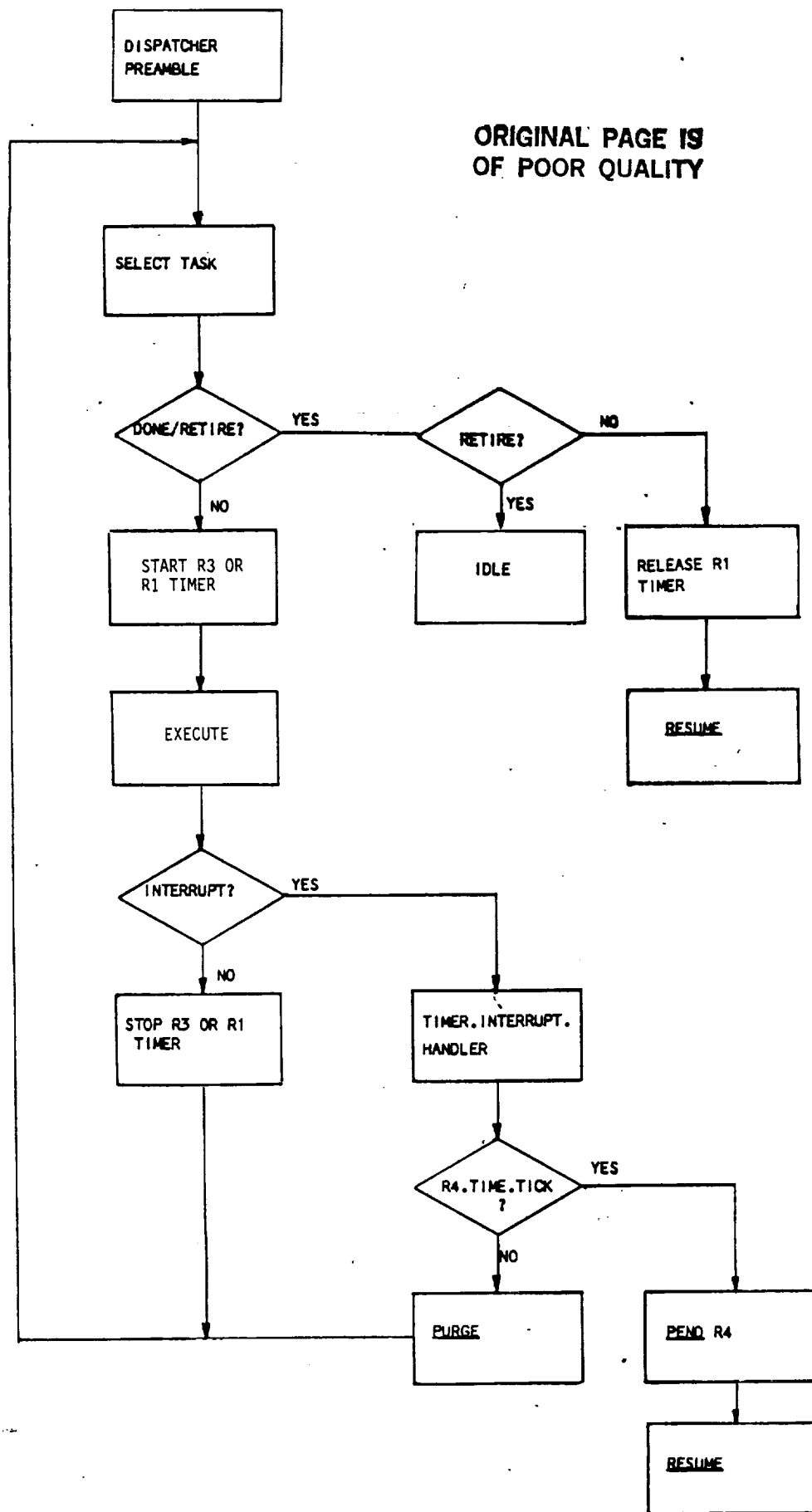
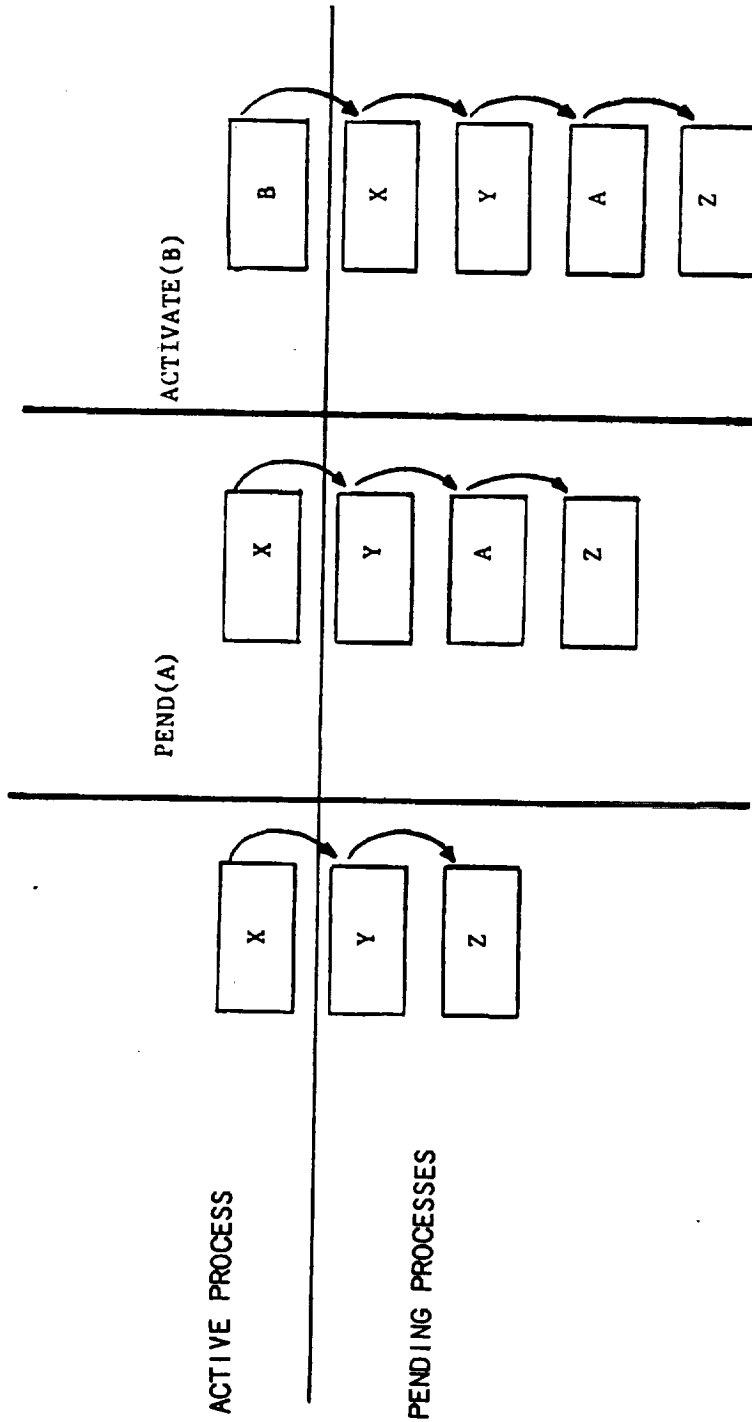


FIGURE 1.5. R3 and R1 Task Invocation Procedure Control Flow

execute in consecutive order. Changes to this order are implemented by the PEND and ACTIVATE routines as depicted in figure 1.6 (adopted from a CSDL briefing chart). Process X is currently being executed by the triad, and is designated as the active process. A PEND(A) command will cause process A to be inserted in the task execution order as shown in the second column in figure 1.6. The ACTIVATE(B) command will immediately transfer control to the process B as shown in the third column of figure 1.6; the previously active process continues at the conclusion of process B.

The response to interrupts is depicted in figure 1.7, adopted from a CSDL briefing chart. Prior to the interrupt, a process denoted as Z is active in the triad, as shown by the "AP" marker. After the interrupt occurs, the address of the process Z PSD is saved in a location of the interrupting process PSD, and the interrupting process becomes active. At the conclusion of the interruption, a RESUME command is executed, and the PSD of process Z becomes active once again.



	B (active)	
	X (pending)	
X (active)	X (active)	
Y (pending)	Y (pending)	
Y (pending)	A (pending)	
Z (pending)		

FIGURE 1.6. Pend and Activate Handling

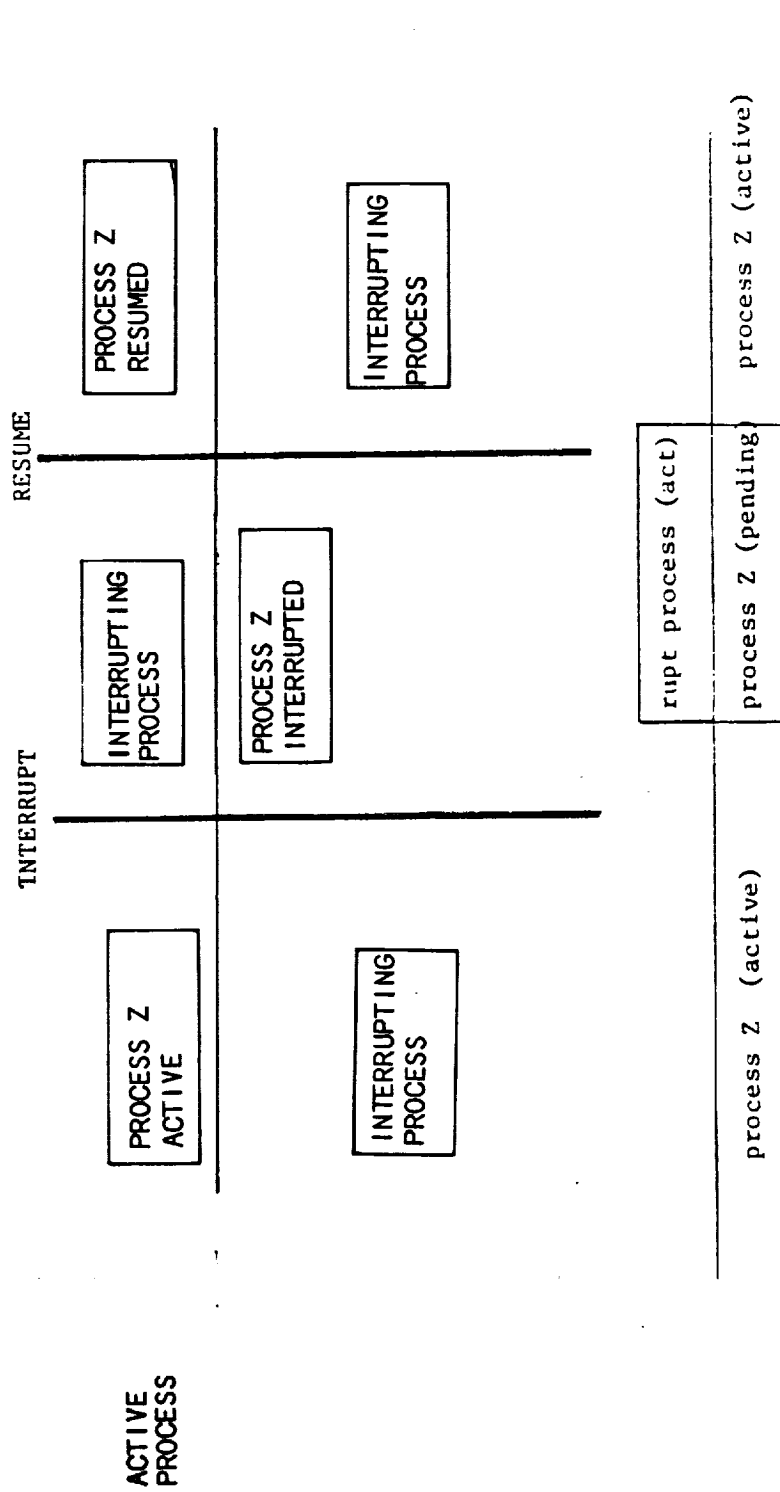


FIGURE 1.7. Interrupt Handling

1.3. COVERAGE OF THE PRIMARY ROUTINE FAILURES

The conception and the design of the acceptance tests are based on the potential failures derived from the description of the dispatcher and associated routines contained in section 1.2. It was found that fault trees aided the systematizing and documentation of the acceptance test development, and therefore, these trees are presented in this section.

Figure 1.8 is the top-level fault tree which shows that failure of any of the five functions associated with all rate group dispatchers (i.e. the five functions of the R4 dispatcher and the task dispatching function of the R3.R1 dispatcher) will result in a failure of the entire dispatcher. Failures of the first three fault categories are expanded in subsequent diagrams referred to in the triangles under the event identifications. Failures in reconfiguration and I/O were not covered in this work. The numbers given in the circles below these events refer to specific section numbers where the relevant acceptance test is described.

The reader should note that any input to an "OR" gate causes its output to be true. Thus, in order for an output fault given at the top of an OR gate to be detected, all inputs must be detected. However, in the case of an "AND" gate, all inputs must be true in order for the output to be true, and thus, only a single fault need be detected in order to assure coverage.

Initialization Faults

A further development of initialization failures is contained in figure 1.9. Two initialization failures would result in dispatcher error conditions: failure to set the triad as R4 Responsible and failure to properly initialize the rate group control blocks.

Because other functions listed in section 1.2.1 under frame count initialization do not necessarily lead to critical failures, they are not considered explicitly. In a portion of the possible range, initialization failures will result in degraded performance which is not sufficiently critical to cause the function to be disabled, and hence, does not warrant invocation of the alternate dispatcher (which itself provides degraded performance relative to the fully functional primary). The following functions were deemed to be in this category under some conditions in restart initialization:

- resetting of error latches, triad busy states, and triad statuses
- initialization of the timer
- lower rate groups set to be executed in later frames
- reconfiguration states are set to 0
- UNLOCK and IPC interrupts are set to 0

Other possible initialization errors could result in conditions more serious than degraded performance. For example, if all triads are set to busy and are unable to restart the R4 processor, then there will be no R4 responsible triad. The ultimate result of these errors is the improper execution of the rate group dispatchers, a condition which is covered by acceptance tests, and hence, no

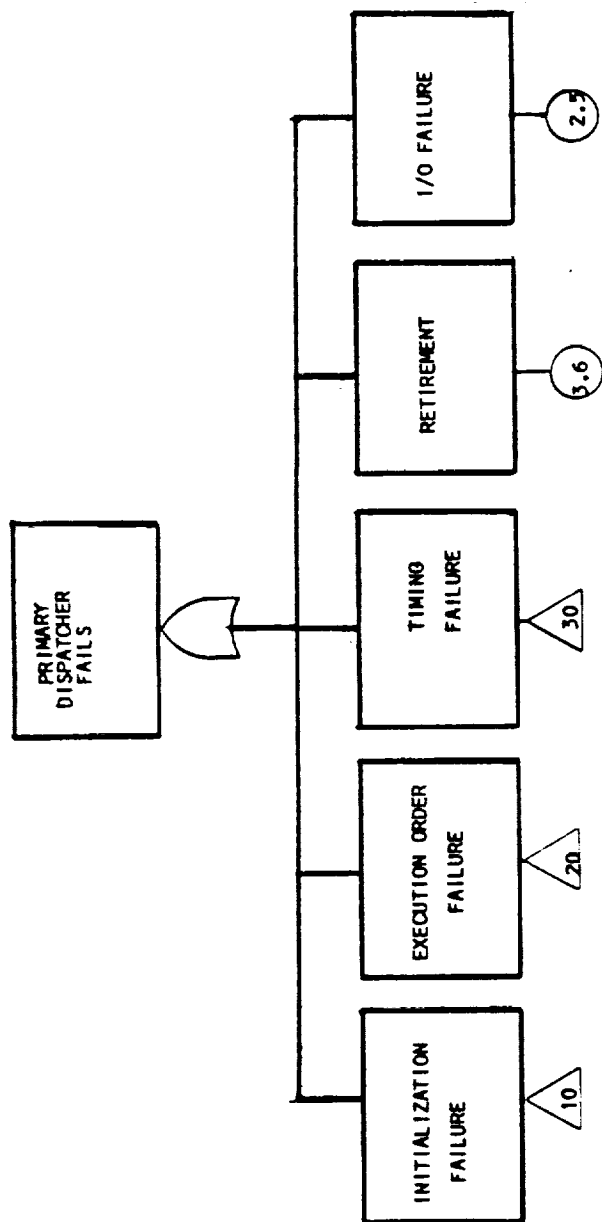


FIGURE 1.8. Top Level Fault Tree for the FTMP Dispatcher

additional provisions are necessary.

Execution Order Failure

This failure class includes errors in frame initiation which causes lower rate group dispatchers not to be invoked at appropriate intervals and errors in task selection and execution which result in failure to dispatch all applications routines of a given rate group.

Figure 1.10 shows the further development of possible failures in this category. Applications tasks could either be omitted, or they could be executed too frequently. Omissions of critical tasks are detected by the functional acceptance tests of section 2, and the setting of variables which result in too frequent executions of tasks are covered by the structural acceptance tests of section 3.

Timing Failures

Failures of the functions described in section 1.2.2 are included in this category. The most apparent source of failures is the hardware clock, which is adequately covered by quadruple redundancy, hardware voting, and spares, and which need not be protected by additional software provisions.

A second cause of failures is the R4 rate group being "stuck", a condition which occurs under the three circumstances shown in figure 1.11. The first circumstance, being stuck in an R4 applications routine is handled by the interval timer acceptance test described in section 2. The second, R4 stuck in task selection group and the third uninterruptible ASM sequence, are covered by a single acceptance test described in section 3.

I/O and Retirement Failures

Because the I/O protocols were not described in detail in ref. 4, it was not possible to devise a set of acceptance tests to provide definitive coverage. However, the relatively powerful "wrap around" acceptance test described in section 2.5 will aid in covering this failure, and based on final design of the I/O procedures, can be incorporated with supporting structural acceptance tests to provide complete coverage.

The reconfiguration strategies have been developed and documented elsewhere, and do not fall under the scope of the dispatcher. However, once the retirement command is given, the dispatcher is responsible for carrying it out. Section 3.6 describes the retirement acceptance test.

Possible Failures Introduced by Recovery Blocks

Figure 1.12 is a fault tree showing the possible failures introduced by the recovery block structure for the dispatcher. As is shown, failures can be due to a primary routine failure coupled with a fault in either the acceptance test or the alternate routine as well as a type II error (i.e. false indication of failure) by the acceptance test and failure of the alternate routine. Coverage of acceptance test failures is handled by both the critical word reset acceptance test, which ensures that a proper critical word mask is used by the dispatcher acceptance test, and the frame count acceptance test, which ensures

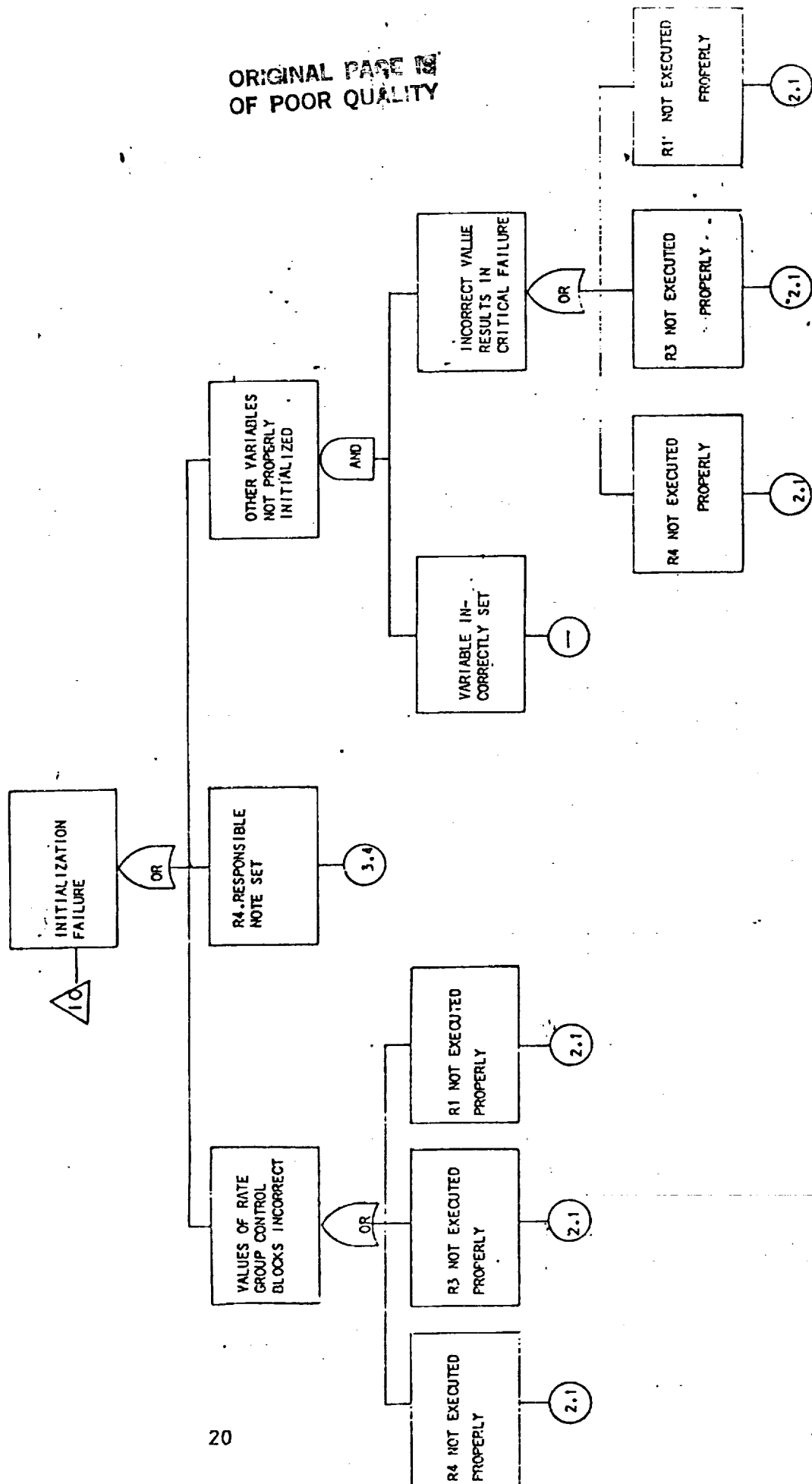


FIGURE 1.9. Initialization Faults

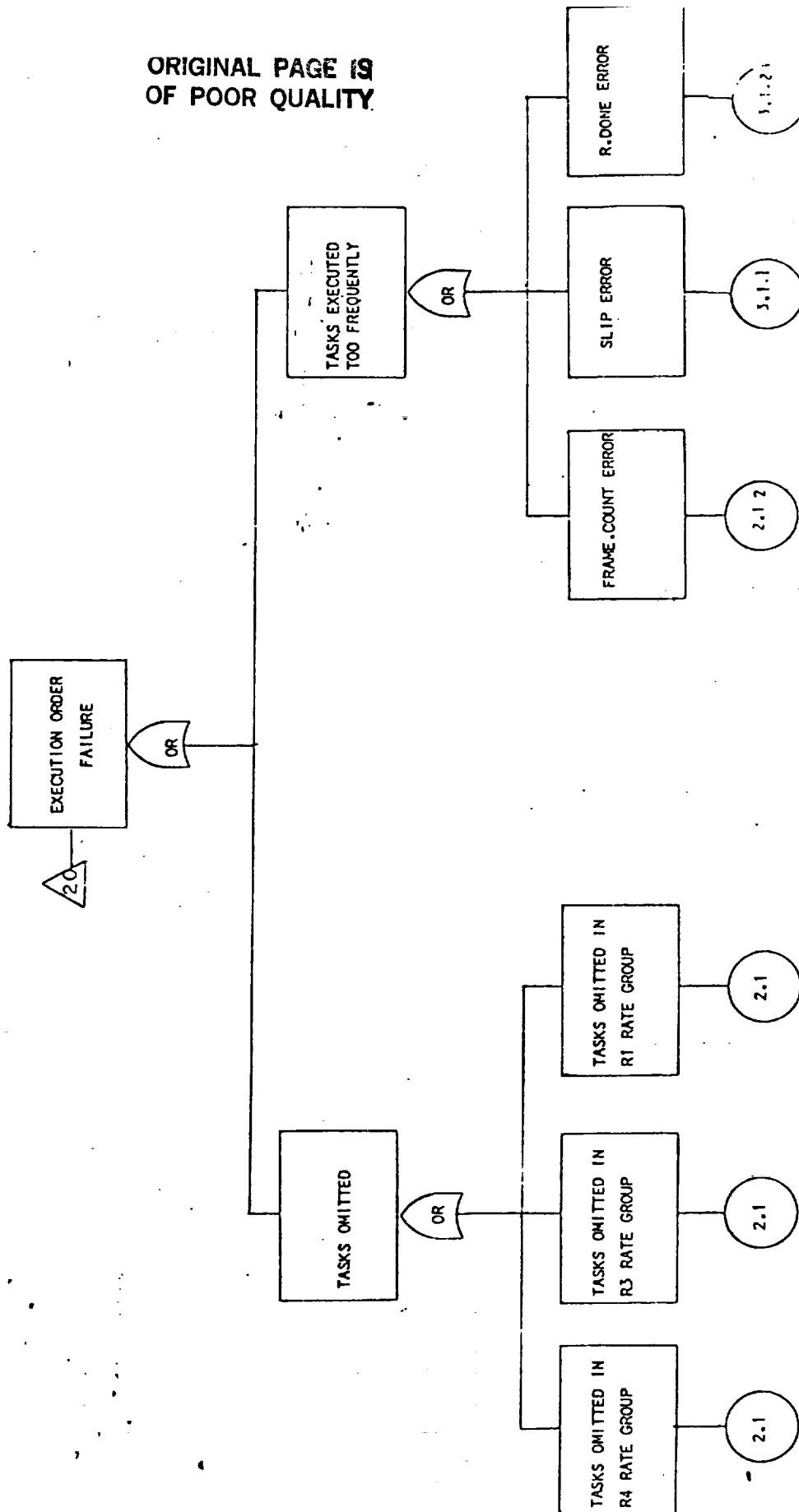


FIGURE 1.10. Execution Order Failures

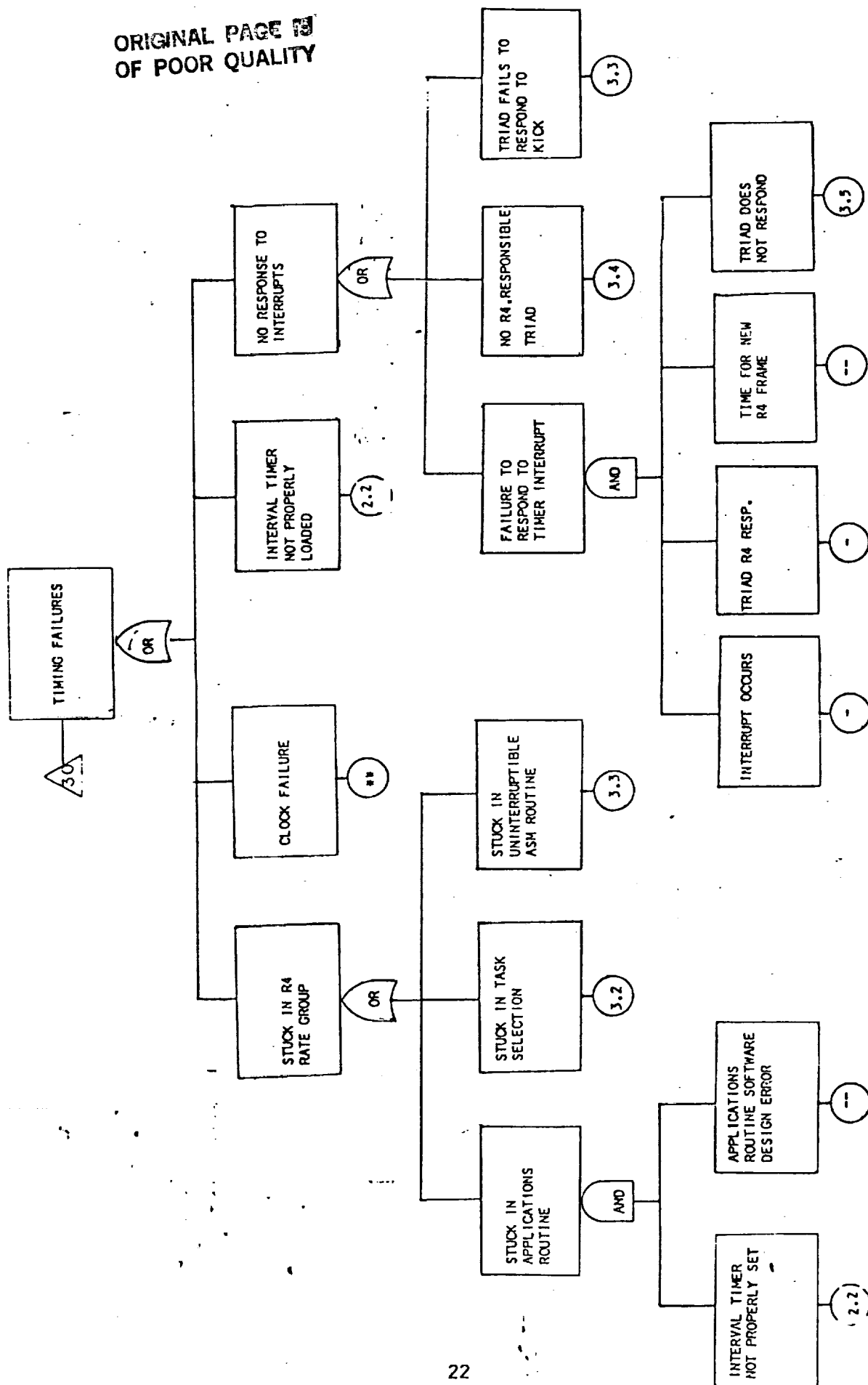


FIGURE 1.11. Timing Failures

ORIGINAL PAGE IS
OF POOR QUALITY

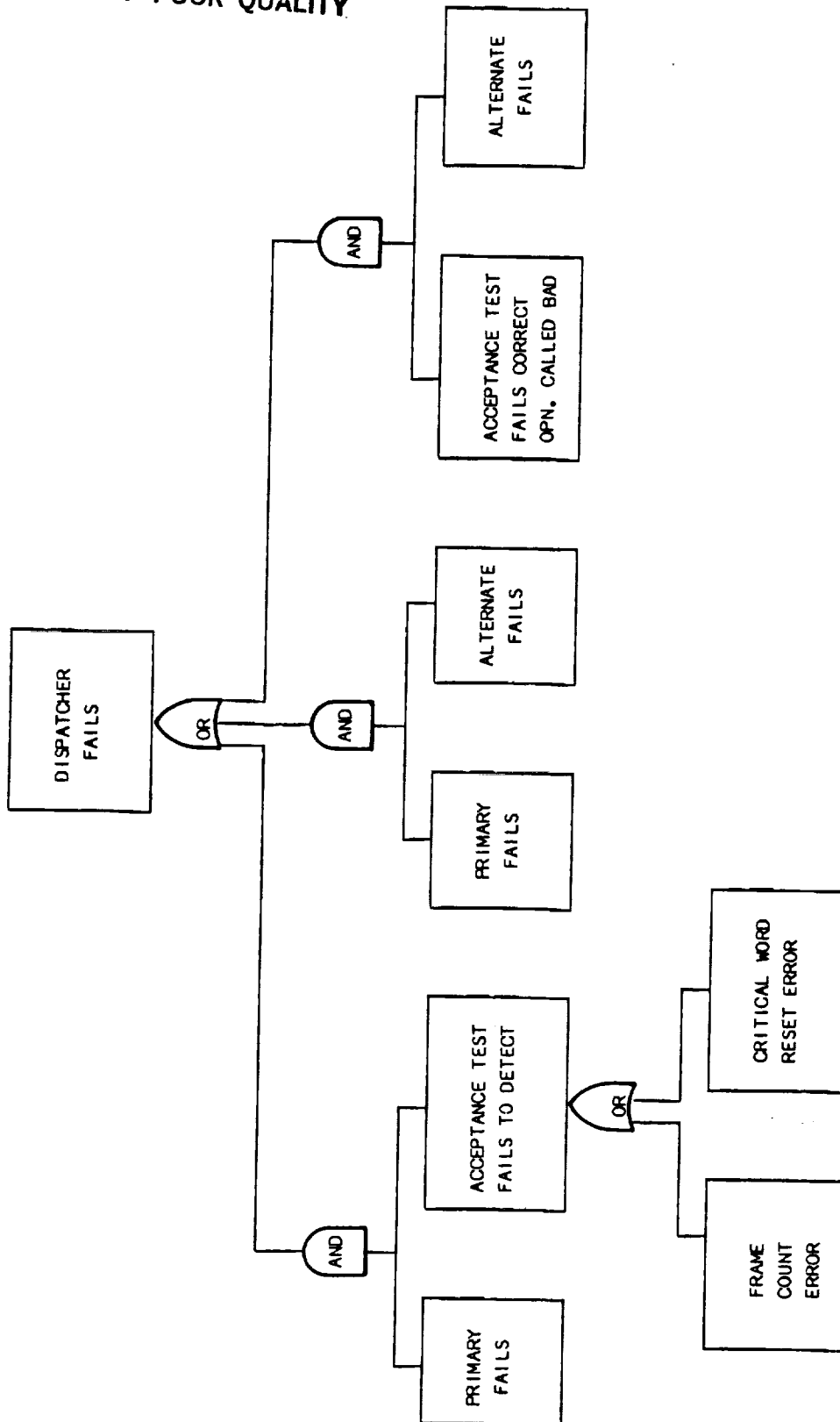


FIGURE 1.12. Recovery Block Faults

that the dispatcher uses the proper frame when testing for the execution of the R1 and R3 rate groups.

1.4. SOFTWARE ERRORS NOT COVERED BY THE DISPATCHER ACCEPTANCE TESTS

The scope of both the functional and structural acceptance tests has been limited to the failures of the dispatcher and supporting routines. AED procedures not covered by the accepted tests include LOCK/UNLOCK, IPC.INTERRUPT, and the reconfiguration tasks.

SECTION 2 - FUNCTIONAL ACCEPTANCE TESTS

As noted in section 1.1, functional acceptance tests are those which test the output of a software module for the achievement of a functional objective. Such tests have been developed for the dispatcher, routines setting the interval timer, and functions associated with I/O.

The dispatcher acceptance test checks memory locations in which critical tasks from each rate group have set bits in the course of their execution. If the words indicate that all critical tasks in the given frame have been run, the dispatcher acceptance test resets the critical words for the next frame. Two additional routines that are associated with this acceptance test check both the input frame count and the output reset critical words. Details of the dispatcher acceptance test are described in section 2.1.

The interval timer acceptance test checks the interval timer after the execution of any routine which can affect the timer value. If the time is less than that of the current frame, a normal exit occurs. Details of the interval timer acceptance test are described in section 2.1.

The I/O acceptance test checks an independent counter showing the number of times the buffer has been accessed and compares it with the frame count. Without a detailed knowledge of the final I/O protocols used for the FTMP, it can not be concluded that this acceptance test provides a definitive determination of normal execution, but it is anticipated that it will be useful, especially if combined with additional specific structural acceptance tests. Details of the I/O acceptance test are described in section 2.1.

Finally, although not defined specifically, each critical applications routine will have its own functional acceptance test which checks the validity of its input and output. Tests for determining whether constraints have been met for the running of applications routines is also part of the function of these acceptance tests.

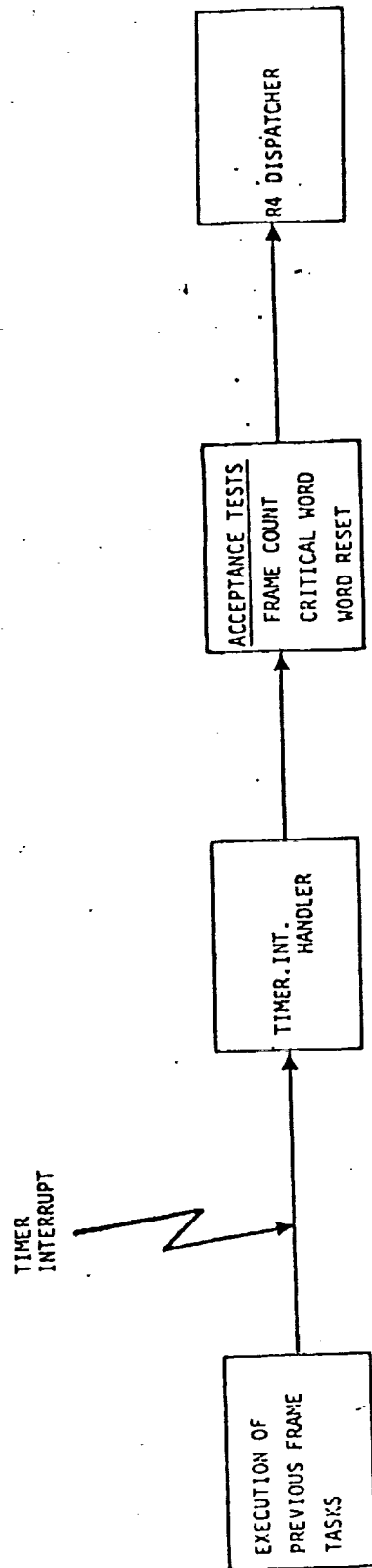


FIGURE 2.1. Sequence of Timer Interrupts and Dispatcher Acceptance Tests

2.1. DISPATCHER ACCEPTANCE TESTS

As noted above, the overall dispatcher acceptance test scheme consists of checking the values of critical words which indicate that tasks crucial to maintaining stable flight conditions have been run in each rate group. However, because the critical task group varies as a function of the frame count, a second test must ensure that the dispatcher uses a proper value for this variable. Finally, the critical words must be reset at the beginning of every frame; the successful execution of this function is verified by a third acceptance test. Thus, a total of three modules for the acceptance tests associated with the dispatcher have been developed:

Test for Failure of the Dispatcher. The overall functional tests for the dispatcher is to determine whether all critical tasks within a given rate group have been executed at the appropriate times. Failures of a number of functions are detected by this test. This test is further described in section 2.1.1.

Test for Failure of the Frame Counter. Failure of the frame counter can result in the improper timing for execution of lower rate groups. If the counter is not incremented or is incremented by greater than one, it is possible that lower rate groups will not execute at all. The test for proper incrementing of the frame counter is described in section 2.1.2.

Critical Word Reset Acceptance Test. Failure to reset the critical word can result in improper assessment of whether the rate groups have been completed. At the conclusion of the dispatcher acceptance test, the critical word is compared with its initial value stored in memory. If these values do not agree, then the alternate scheduler-dispatcher is invoked. This test is further described in section 2.1.3.

The point in the frame at which these tests are executed is shown in figure 2.1. After the R4.RESPONSIBLE triad receives an interrupt signaling the beginning of a new frame, the acceptance tests are invoked, and the R4 dispatcher is restarted.

2.1.1. Dispatcher Critical Word Acceptance Test

The specification of the dispatcher requires that all R4 tasks will be completed every frame, all R3 tasks every second frame, and all R1 tasks every eighth frame. The acceptance test will rely on the existing clock and frame count to verify that this functional requirement is met. However, an independent acceptance test will verify the frame count by means of two independent counters (see sec. 2.1.2). The acceptance test will check a critical word for the appropriate rate group. If this word indicates that all tasks have been completed (by being set to all 1's), then it will reset the word (in a manner that will also be fault-tolerant, see sec. 2.1.3) and proceed to test the next lower rate group as appropriate. If a discrepancy in the critical word is detected, the acceptance test will invoke the alternate dispatcher.

Figure 2.2 shows the dispatcher and accompanying fault tolerant provisions (this configuration is the same for all rate groups) invoked at the beginning of a new frame. If the dispatcher acceptance test is satisfactory, the primary dispatcher is re-executed as shown in the left-hand branch of figure 2.2, and if not, the alternate dispatcher is executed. Critical application's routines will set appropriate bits in the critical word for that rate group.

Figure 2.3 is the Nassi-Schneideman diagram of the dispatcher acceptance test and table 2.1 shows the requirements. As each rate group critical word is tested, it is reset, and the reset function in turn is tested (section 2.1.3). If any rate group critical word is not satisfactory, the alternate dispatcher is invoked.

2.1.2. Frame Count Acceptance Test Requirements

The frame count acceptance test independently verifies that the frame counter has been properly incremented. It is invoked prior to execution of the dispatcher as shown in figure 2.1. Figure 2.4 is an Nassi-Schneideman diagram for this acceptance test. As shown in figure 2.5, two independent frame counters, NEW.FRAME and OLD.FRAME (each counting from 0 to 15) in this acceptance test are used to ensure that proper incrementing has been carried out within the test. A comparison is then made with the dispatcher frame counting variable FRAME.COUNT. If a discrepancy is found, all frame counts are restarted at 0 without the need for the alternate scheduler-dispatcher. However, if a discrepancy persists for an arbitrary number of consecutive times (we have chosen three consecutive times), then the acceptance test invokes the alternate scheduler-dispatcher. The requirements for the frame count acceptance test are given in table 2.2

2.1.3. The Word Reset Acceptance Test

If the critical words are not properly reset, the dispatcher critical word acceptance test may fail to detect that the dispatcher has not invoked all applications tasks. Most likely, the failure of the critical word reset will result in the final word (at the end of the frame) indicating that all critical tasks have not been completed when they all have actually been run. However, there is also the possibility that the critical word is not properly reset and that the tasks are not completely executed. The dispatcher critical word acceptance test will then indicate proper completion even though this has not been achieved.

The requirements proposed for the word reset acceptance test are given in table 2.3, and figure 2.6 shows the Nassi-Schneideman diagram.

TABLE 2.1. Dispatcher Critical Word Acceptance Test Requirements

1. The dispatcher critical word acceptance test checks rate group critical words at the beginnings of the appropriate frames after the frame count has been verified by an independent acceptance test.
 - a. The R4 critical word is checked at the beginning of every frame.
 - b. The R3 critical word is checked at the beginning of even frames (i.e. 0, 2, 4, 6, 8, 10, 12, and 14).
 - c. The R1 critical word is checked at the beginning of frame 0 and 8.
2. If the rate group critical word is correct, the dispatcher acceptance test will reset it. A separate acceptance test verifies its re-initialization.
3. If the rate group critical word is not correct, the dispatcher acceptance test will invoke the alternate scheduler/dispatcher.

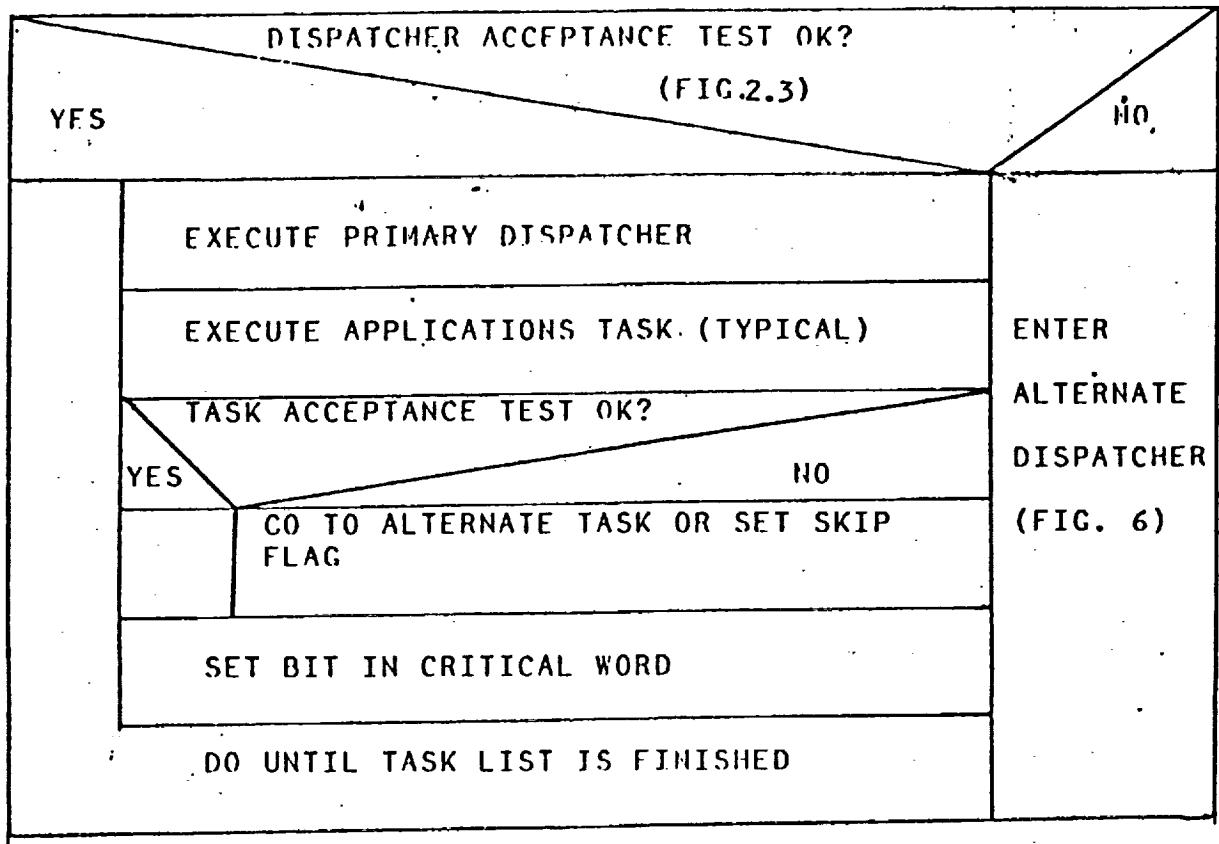


FIGURE 2.2. Fault Tolerant Provisions for the FTMP Dispatcher

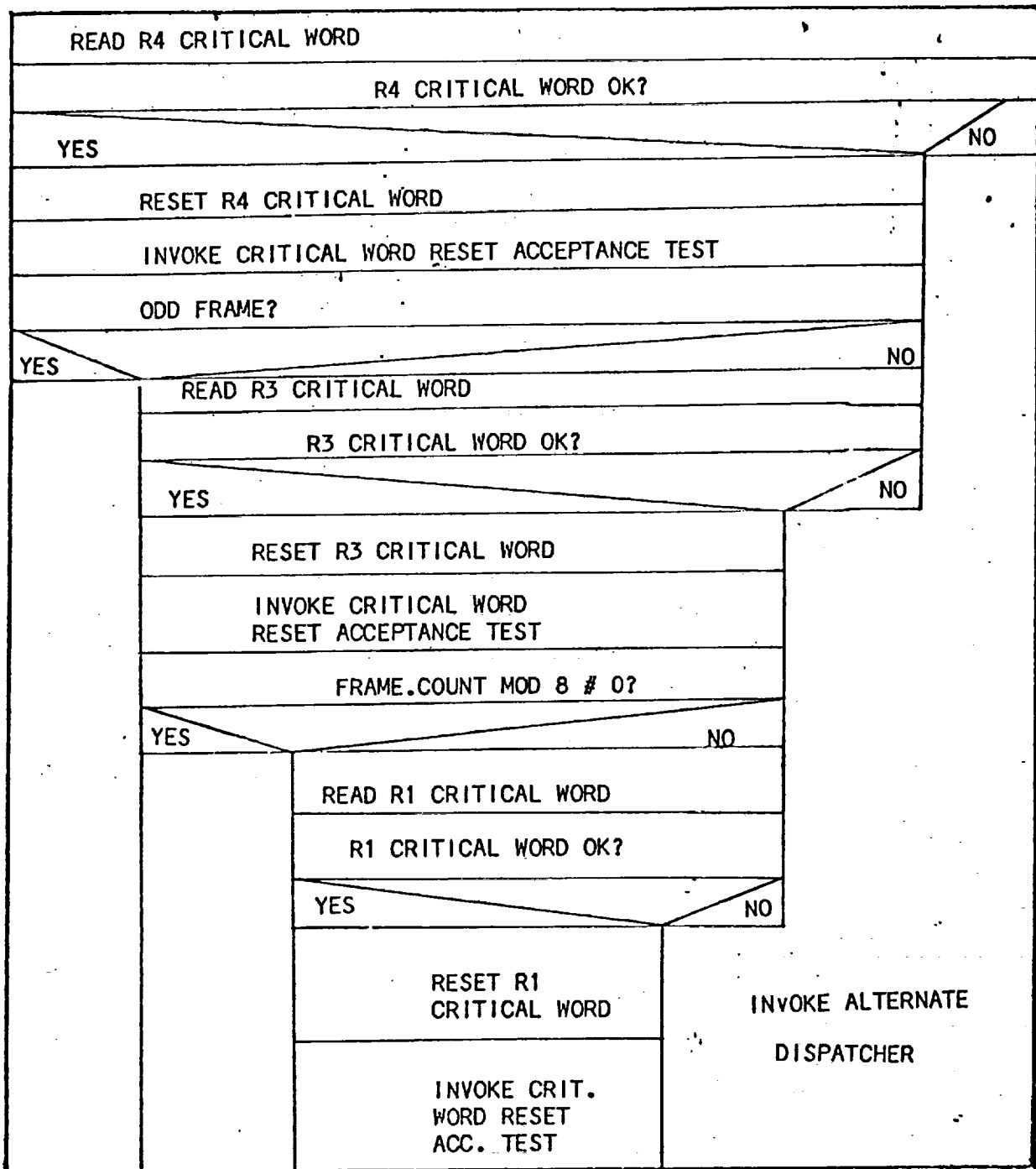


FIGURE 2.3. Dispatcher Critical Word Acceptance Test Module

TABLE 2.2. Frame Count Acceptance Test Requirements

1. The frame count acceptance test will be invoked every frame as an R4 critical applications routine.
 - a. The frame count acceptance test will set a bit in the R4 critical word.
 - b. The dispatcher critical word acceptance test will check the R4 critical word to verify that this acceptance test has been run in the previous frame.
2. The frame count acceptance test will increment its own independent frame counter, and then ensure that it has been properly incremented.
 - a. The acceptance test frame counter, NEW.FRAME, will be incremented in the range 0 to 15 (by using NEW.FRAME modulo 16).
 - b. The difference between NEW.FRAME and a second acceptance test counter, OLD.FRAME will be checked.

If the difference is 1, then OLD.FRAME is set equal to NEW.FRAME (i.e. NEW.FRAME is properly incremented)

If the difference is not equal to 1, the NEW.FRAME, OLD.FRAME, and FRAME.COUNT (the primary dispatcher frame counter) are set to 15 so that tasks for all three rate groups will be executed in the subsequent frame. An error counter, FRAME.FAIL.COUNTER, is incremented and checked to see that it has not reached a preset limit (3 is chosen at present). If FRAME.FAIL.COUNTER has exceeded the limit, then the alternate scheduler-dispatcher is called. If it has not, then the primary dispatcher is invoked at the new frame count.
3. If the acceptance test frame counter has been properly incremented, it is compared to the primary frame counter.
 - a. If the acceptance test frame counter and the primary frame counter agree, FRAME.FAIL.COUNTER is set to zero and the primary dispatcher is invoked.
 - b. If the primary frame counter and the acceptance test frame counter do not agree, NEW.FRAME, OLD.FRAME, AND FRAME.COUNT are set to 15 so that all three rate groups will be executed in the subsequent frame. FRAME.FAIL.COUNTER is incremented and checked to see that it has not reached a preset limit (3 is chosen at present). If FRAME.FAIL.COUNTER has exceeded the limit, then the alternate dispatcher is called. If it has not, then the primary dispatcher is invoked at the new frame count.

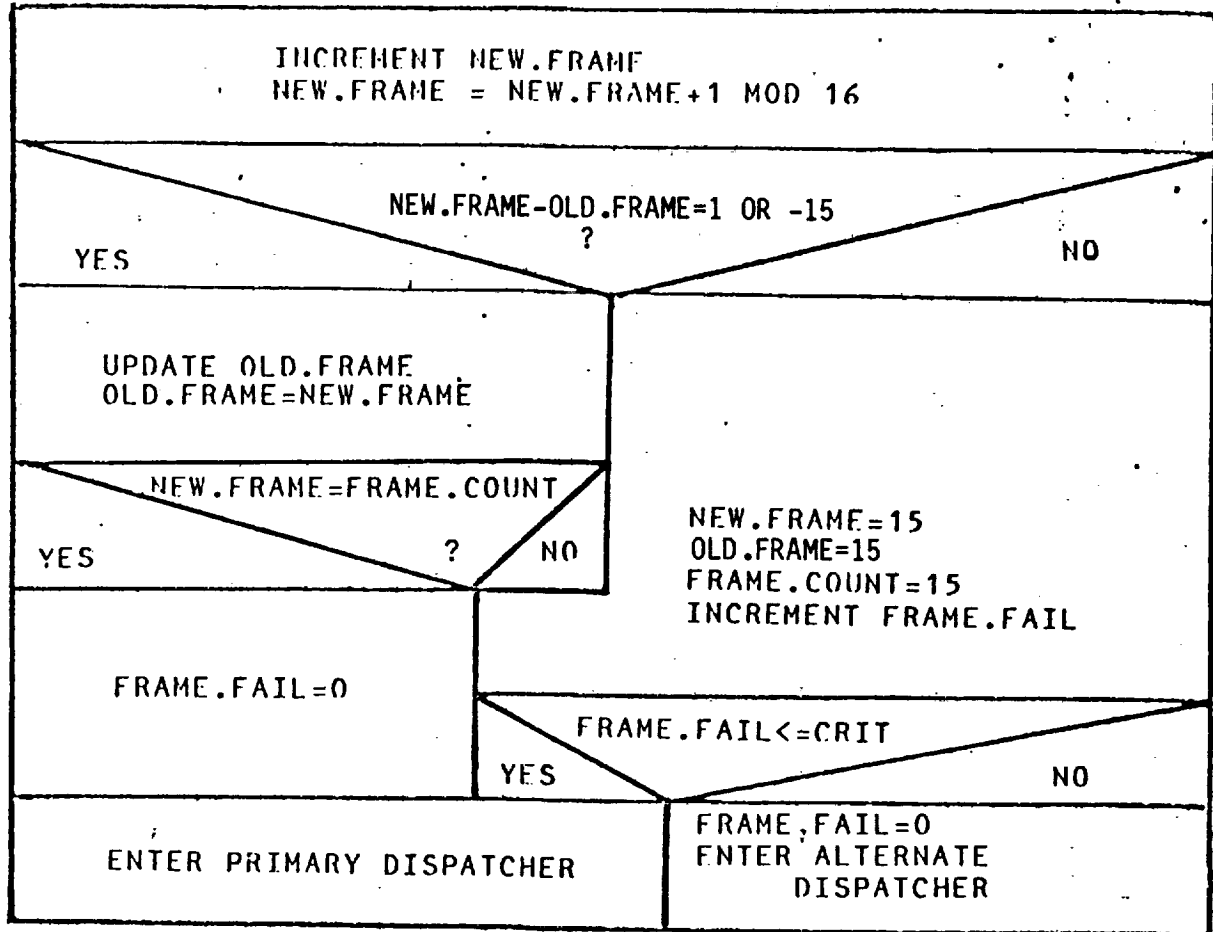


FIGURE 2.4. Frame Count Acceptance Test Module

TABLE 2.3. Critical Word Reset Acceptance Test Requirements

1. The critical word reset test is called by the dispatcher acceptance test.
2. The memory location containing the rate group critical word which has just been reset is compared with the memory location containing the initial value to which the word should have been changed.
3. If these values agree, normal execution of the primary dispatcher is continued.
4. If these values do not agree, then the alternate dispatcher is invoked.

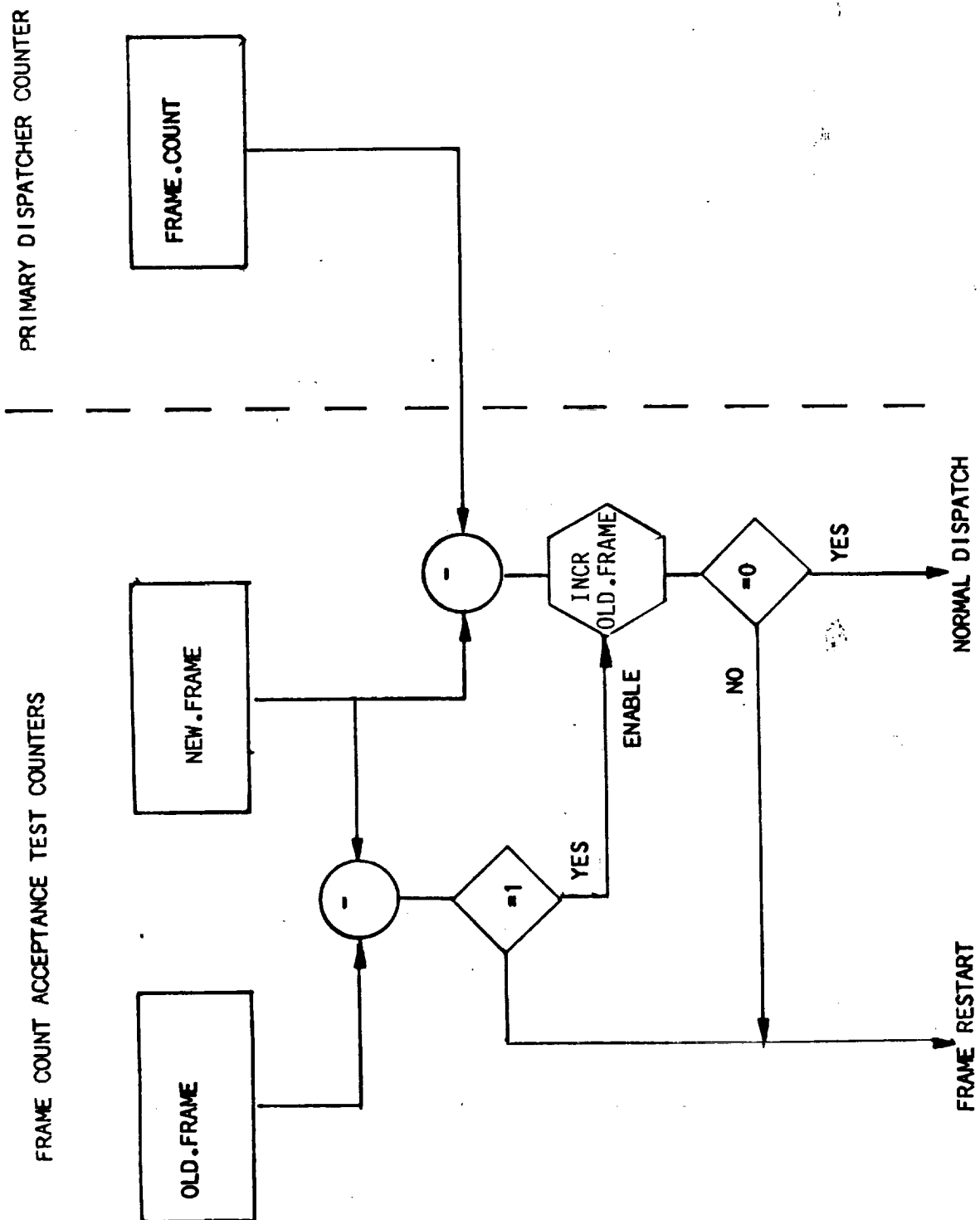


FIGURE 2.5. Frame Counters

CRITICAL WORD = INITIAL VALUE ?	
YES	NO
CONTINUE NORMAL EXECUTION	INVOKE ALTERNATE DISPATCHER

FIGURE 2.6. Critical Word Reset Acceptance Test Module

2.2. INTERVAL TIMER ACCEPTANCE TEST

The interval timer is the crucial system component which ensures that interrupts occur at the proper interval. A total of ten AED procedures control and arm the interval timer:

```
HOLD.R3.R1.TIMERS
HOLD.R1.TIMERS
RELEASE.R3.R1.TIMERS
RELEASE.R1.TIMER
START.R4.TIMER
START.R3.TIMER
START.R1.TIMER
STOP.R4.TIMER
STOP.R3.TIMER
STOP.R1.TIMER
```

Each of these procedures must determine the time to the next interrupt, the type of interrupt, load the interval timer register with the appropriate value, and arm it. Failure to load, arm, or properly identify the next interrupt will not be detected by any of the previously defined acceptance tests.

A means of detecting these failures is therefore included as an additional functional acceptance test. This test, designated as the Interval Timer Acceptance Test, ensures that the value in the interval timer of the R4 responsible triad is less than the value to the next R4 frame as defined by the difference between R4.TICK.TIME and TIME.NOW. If the interval is greater than the difference, or if TIME.NOW is greater than the R4.TICK.TIME, then a functional error has occurred in one of the timer routines, and the alternate dispatcher is invoked. Table 2.4 lists the requirements for the Interval Timer Acceptance test, and figure 2.7 is the Nassi-Schneideman diagram for this procedure.

The reader should note that this test will not affect the values of non-R4 responsible timers. These intervals continue to extend beyond the next R4 frame, and can not be used as back-ups to ensure the timely start of the R4 frame.

Although the acceptance test does not check for the occurrence of the appropriate interrupt or that R4.TICK.TIME has been set to a new value explicitly, it does so implicitly through the condition identified in 4b of table 2.4. If an interrupt other than that for a new frame has been set at the R4 interrupt time, then the acceptance test will detect the failure by noting that the R4.RESPONSIBLE flag is still set (and hence, that no triads are working on the R4 rate groups). This same check can be used to ensure that a new R4.TICK.TIME value has been entered. If the difference between R4.TICK.TIME and TIME.NOW at the end of the R4 task execution cycle is less than zero (i.e. was not reset in the R4 Dispatcher), then the failure is also detected.

TABLE 2.4 REQUIREMENTS FOR THE INTERVAL TIMER ACCEPTANCE TEST

1. The Interval Timer Acceptance Test shall be invoked after each of the following routines is called:

HOLD.R3.R1.TIMERS
HOLD.R1.TIMERS
RELEASE.R3.R1.TIMERS
RELEASE.R1.TIMER
START.R4.TIMER
START.R3.TIMER
START.R1.TIMER
STOP.R4.TIMER
STOP.R3.TIMER
STOP.R1.TIMER

2. If the triad is not R4.RESPONSIBLE, and the R4 dispatcher is not active, the acceptance test will return control to the calling routine.
3. If the R4 dispatcher is active, the acceptance test will compare the value of the interval time with the maximum permissible R4 limit. If the value of the limit is exceeded, the acceptance test will invoke the alternate dispatcher.
4. If the triad is R4.RESPONSIBLE, the acceptance test will invoke the alternate dispatcher if either of the following conditions is met:
 - a. $R4.TICK.TIME - TIME.NOW < INTERVAL.TIMER$
 - b. $R4.TICK.TIME - TIME.NOW < 0$
5. If neither of the conditions of (4) are met, then the acceptance test will perform a normal exit and return control to the calling routine.

IS TRIAD R4.RESPONSIBLE ?			
YES			NO
READ INTERVALTIMER		R4.ACTIVE = TRUE ?	
		YES	NO
INTERVAL TIMER > (R4.TICK.TIME - TIME.NOW) OR R4.TICK.TIME - TIME.NOW < 0		INTERVAL TIMER > MAX.R4.TIME	
YES	?	YES	NO
INVOKE ALTERNATE		INVOKE ALTERNATE	NIL
		NIL	NIL

FIGURE 2.7. Interval Timer Acceptance Test

2.3. INPUT/OUTPUT ACCEPTANCE TESTS

In addition to the acceptance tests dealing with execution of the dispatcher, the input/output routines must also be tested. The test criterion is the number of times that a data buffer is accessed in a given time period, and this is compared with the frame count appropriately adjusted for even and odd frames as defined in the requirements shown in table 2.5.

Because the I/O procedures for the MIL-STD 1553 bus were not within the scope of this work, no specific recommendations will be made. However, some general comments on the nature of the test are described here.

The I/O acceptance tests verify that the dispatcher has invoked the input/output routines in the proper sequence, and that the upcoming values which are used by the applications tasks are reasonable relative to those which had been used in the previous cycle. As currently conceived, the dispatcher reads in the data for all rate groups to be executed in a given frame during the R4 frame initiation from a buffer location to which data is constantly being transmitted. The procedures performing this task, RX.IN() and RX.OUT(), are executed as part of the R4 dispatcher.

There are two major acceptance tests: (1) a test invoked by the dispatcher to determine that the buffers of each of the data group have been accessed at the appropriate rates (including an even/odd test), and (2) a test for the reasonableness of each data point, i.e. that the point varies in a reasonable way from (e.g. is it within a certain range of) the previous point.

In order to verify that the data group access counters have been checked, the I/O acceptance test must be an R4 critical task (whose execution is noted in the critical word). The requirements for this acceptance test are given in table 2.5.

TABLE 2.5. I/O ACCEPTANCE TEST REQUIREMENTS

1. The number of times that a data group has been accessed will be compared with the frame counts.
 - a. Data groups associated with R4 tasks will have counters that will be compared with R.FRAME.(RG4).

If the data group counter equals R.FRAME.(RG4), then the counter will be incremented.

If the data group counter is greater than 15, then the counter will be reset (to modulo 16) prior to being compared with the R.FRAME.(RG4) variable.

If the data group counter does not equal R.FRAME.(RG4), the alternate dispatcher will be called.
 - b. Data groups associated with R3 tasks will have counters that will be compared with R.FRAME(RG3).

If the data group counter equals R.FRAME(RG3), then the counter will be incremented.

If the data group counter is greater than 8, then the counter will be reset (to modulo 8) prior to being compared with R.FRAME(RG3).

If the data group counter does not equal R.FRAME(RG3), then the alternate dispatcher will be called.
 - c. Data groups associated with R1 tasks will have counters that will be compared with R.FRAME(RG1).

If the data group counter equals R.FRAME(RG1), then the counter will be incremented.

If the data group counter is greater than 2, then the counter will be reset (to modulo 2) prior to being compared with R.FRAME(RG1).

If the data group counter does not equal R.FRAME(RG1), then the alternate dispatcher will be called.
2. Within each applications task, a data acceptance test will determine whether the input is reasonable (i.e. is within an acceptable range of the previous value).
 - a. If the data is not reasonable, the alternate applications task will be invoked

OR

- b. If the data is not reasonable, then the applications task will use a backup data module.

(the use of either alternative may be dependent on the specific nature of each applications task, flight conditions, and general practicality)

- 3. Each rate group will have an acceptance test to ensure that the appropriate (even or odd) buffer is being accessed.
- 4. The I/O acceptance tests will be R4 critical tasks.

2.4. APPLICATIONS ROUTINES

It is assumed that each of the applications routines will have its own acceptance test to verify correct execution. However, in addition to testing for the validity of output, these acceptance tests will perform two additional tests on the operation of the dispatcher: that constraints have been met and that the critical word bit has been set for this routine.

The setting of critical word bits will occur at the beginning of each applications routine, and a function of the applications task acceptance test is to ensure the appropriate value of the critical word at the conclusion of task execution.

Testing for the violation of constraints is most efficiently handled on the applications task level as part of the check for the validity of input. However, failure to meet constraints is a dispatcher fault and should result in invocation of the alternate dispatcher if such a failure has critical implications. Thus, in the event that constraints have not been met, the applications routine will reset its bit in the critical word to indicate that it has not run. When the dispatcher functional acceptance tests are run, they will detect failure to execute a critical task, and will invoke the alternate dispatcher.

SECTION 3 - STRUCTURAL ACCEPTANCE TESTS

Structural acceptance tests check sections of code to ensure that key variables have been set and functions have been executed. In section 1, the coverage of these tests is shown to be particularly important. Specific applications of structural acceptance tests are described for the following modules or subfunctions of the dispatcher:

SLIP and R.DONE - Because the functional acceptance test requires both variables as criteria for deciding whether to read the critical word, errors in these variables as well as errors in the critical word will not be covered without assurance of the correctness of these inputs. Two acceptance tests described in section 3.1 detect these failures.

STUCK IN R4 FRAME - This condition can lead to the delaying of the next R4 frame initiation to such an extent that critical tasks will not be executed at their design rates. Failure to complete the current R4 frame in timely manner is detected by the acceptance test described in section 3.2.

KICK - If an R4.RESPONSIBLE triad retires without designating another triad to restart the new frame, then the entire task scheduling and dispatching function will be lost with no indication of failure by the acceptance tests, which will not have been invoked. An acceptance test verifying that KICK has been successfully executed is described in section 3.3.

R4.RESPONSIBLE - A related task is the ensuring that one and only one triad is R4.RESPONSIBLE. Failure of the R4.RESPONSIBLE triad to invoke the new frame will be detected by the Interval Timer acceptance test described in section 3.4.

UNINTERRUPTIBLE CODE - Sections of uninterruptible ASM code in the FTMP operating system may lead to infinite loops which would ultimately result in failure to restart the R4 frame if executed by the R4.RESPONSIBLE triad. The uninterruptible code acceptance test described in section 3.5 covers this error.

RETIREMENT - Failure of a triad to properly respond to a retirement command can result in a pathological condition with severe system consequences. This condition will be detected by the retirement acceptance test described in section 3.6.

3.1. ERRORS IN SLIP AND R.DONE

The Dispatcher Acceptance Test will not detect errors in the setting of SLIP and R.DONE for lower rate groups. As a result, two structural acceptance tests are necessary in order to ensure that these variables are properly set. They must detect the following failures:

(1) the failure of lower rate groups to complete the iterations within a certain time period manifested by exceeding a minimum value on SLIP, i.e. the most negative value it may assume (which may be altered in the course of flight), and

(2) the detection of failure of a rate group to be marked as complete as it finishes its task list.

A third failure, setting RX.DONE to TRUE when the rate group has not completed execution will be detected by the functional acceptance test of the dispatcher.

3.1.1. Maximum Absolute Value for SLIP

This test, whose requirements and N-S diagram are shown in table 3.1 and figure 3.1, compares SLIP with a limit set at the maximum tolerable schedule slippage rate for each flight mode. This limit, designated as MINSLIP, is assigned a value during the dispatcher initialization, and may be altered in the course of the flight. MINSLIP is changed by an applications routine which has an associated acceptance test.

An additional failure, the decrementing of SLIP when the lower rate group has completed its task list, is also covered by this test. If this decrementing is more frequent than allowed for by MINSLIP, the alternate dispatcher will be invoked. An implicit assumption of this scheme is that the degraded execution mode resulting from the too frequent execution of lower rate groups is preferable to the invocation of the alternate dispatcher.

3.1.2. RX.DONE not set to TRUE when task list execution completed

This test will compare the first task in the rate group task list, contained in the the array CONTROL location 0, with the next task to be executed by the rate group, contained in CONTROL(1). If these two are the same, it implies that the previous iteration has been completed, and R.DONE for this rate group should be set to TRUE. If not, the alternate dispatcher is invoked. The requirements for this acceptance test are shown in table 3.2, and the corresponding N-S diagram is shown in figure 3.2.

TABLE 3.1. Requirements for the SLIP Acceptance Test

1. The SLIP value acceptance test is invoked by the R4 dispatcher at the beginning of each frame.
2. The absolute value of SLIP is compared to its maximum limit. If exceeded, the alternate dispatcher is invoked.
3. If SLIP is greater than zero, the alternate dispatcher is invoked.
4. The value of MINSLIP is set as part of the initialization section of the R4 dispatcher. It is altered at the appropriate flight mode changes by a critical applications routine with an associated acceptance test.

TABLE 3.2. Requirements for the RX.DONE Acceptance Test

1. The RX.DONE acceptance test is invoked by the R4 dispatcher at the beginning of each frame.
2. The values of the two elements in the array CONTROL of the lower rate group PCB'S are compared. The first element is the pointer to the first applications routine of the lower rate group, and the second element is the pointer to the next task to be executed when the dispatcher is called.
3. If the value in CONTROL(1) is null, and the RX.DONE variable for that rate group is FALSE, the alternate dispatcher is invoked.

SLIP < MINSLIP ?	
YES	NO
INVOKE ALTERNATE DISPATCHER	NIL

FIGURE 3.1. SLIP Acceptance Test

RX.CONTROL(1)=NULL AND R.DONE=FALSE OR RX.CONTROL(0)=RX.CONTROL(1) ?	
YES	NO
INVOKE ALTERNATE DISPATCHER	NIL

FIGURE 3.2. R.DONE Acceptance Test

3.2. STUCK IN R4 ACCEPTANCE TEST

Acceptance tests described in this section and in section 2 serve to ensure that the R4 dispatcher will be re-entered at the start of a new frame under all circumstances. However, if the R4 tasks are being dispatched late or some other disorder exists within the R4 dispatcher, then no error condition will be detected. The purpose of the STUCK IN R4 acceptance test is to ensure that if there is a delay in the dispatching of the R4 tasks, the delay is within some previously defined acceptable limits.

Requirements for the test are shown in table 3.3, and figure 3.3 is the corresponding N-S diagram. The test uses an applications routine selection counter, R4.APP.COUNTER, which is initialized at the beginning of each R4 frame and is incremented immediately after the successful selection of an R4 applications task. The expected completion time for the R4 frame is determined by the difference between TIME.NOW and R4.TICK.TIME, the time at which the present R4 frame was started. If this difference is greater than R4.PERIOD, then R4.APP.COUNTER is compared to a limit denoted as R4.LATE.LIM, which is the minimum number of tasks expected to be executed by the end of the frame. If R4.APP.COUNTER is less than R4.LATE.LIM, the alternate dispatcher is invoked.

R4.LATE.LIM can be dynamically changed during subsequent cycles of the R4 task selection and dispatching by means of a linear (or other function) in order to ensure that satisfactory progress is made on completion of the R4 iteration after the expected completion time. The merits of monitoring progress must be weighed against the disadvantages in invoking the alternate dispatcher when the situation is not sufficiently critical to warrant such action.

Table 3.3. Requirements for the STUCK IN R4 acceptance test

1. The STUCK IN R4 acceptance test will be invoked after the execution of procedure SELECT.TASK.
2. An R4 applications routine counter, R4.APP.COUNTER, will be initialized at the start of each R4 frame. After the successful selection of a task, R4.APP.COUNTER will be incremented.
3. If a task has been successfully selected, the acceptance test will check the difference between the current time, TIME.NOW, and the time since the R4 dispatcher was re-started (R4.TICK.TIME).
 - a. If this difference is less than R4.PERIOD, the acceptance test will exit without further executable statements.
 - b. If this difference is greater than R4.PERIOD, R4.APP.COUNTER will be compared with R4.LATE.LIM. If R4.APP.COUNTER is less than R4.LATE.LIM, the alternate dispatcher will be invoked.
4. R4.LATE.LIM may be changed dynamically if monitoring the progress of the applications task following the expected end of the frame is critical.

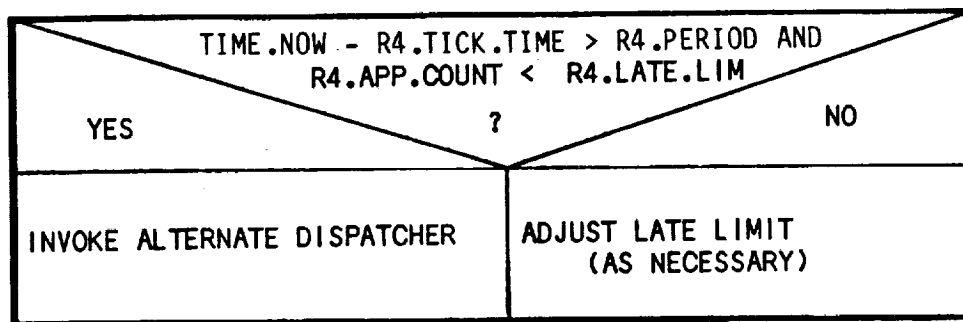


FIGURE 3.3. STUCK.IN.R4 Acceptance Test

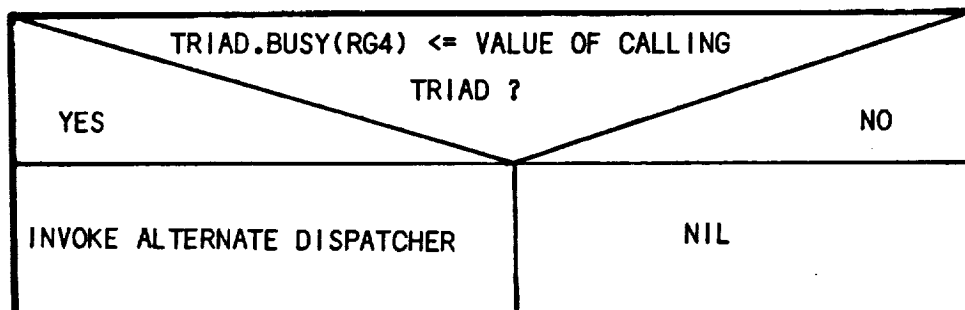


FIGURE 3.4. KICK Acceptance Test

3.3. KICK ACCEPTANCE TEST AND MODIFICATIONS TO KICK PROCEDURES

The AED procedure KICK performs two functions in the R4 dispatcher: (1) to restart the R4 rate group in other triads once the R4.RESPONSIBLE triad has performed the requisite initializations, and (2) to transfer the R4 rate group to another triad in order to ensure that it becomes R4 responsible should the first triad be ordered to retire.

The first function is not deemed to be critical; if the R4 dispatcher does not successfully "wake up" the other triads, the event itself will not always cause a fatal system failure. On the other hand, the failure of the second function will result in there being no R4 responsible triad, and no other indication of the failure with the result that there will be no response to the start of a new R4 frame.

In the design of the KICK routine as described in Ref. 4, the "kicking" triad searches for another eligible triad to which to transfer the R4 dispatcher. If none is found, a normal exit occurs, but the dispatcher is not transferred. This design is adequate for the first purpose described above, but not for transfer of the dispatcher prior to retirement. It is therefore necessary to construct a second procedure which will invoke the alternate dispatcher if no other triads are available for the R4 dispatcher.

This second procedure, designated as KICK2, is identical to the original design of KICK with a single exception, the addition of an AED statement at the end of the procedure. This statement will cause control of the triad to be passed to the alternate dispatcher should there be no other triads available for taking the restart responsibility. Such a statement is not desirable for the transfer of the R4 dispatcher as part of the normal triad execution. The reader should note that this statement will be executed only under the following conditions:

- (1) the last triad executing the R4 dispatcher receives a RETIRE command from the configuration controller, and
- (2) no other triad will receive the R4 dispatcher in order to set its own R4.RESPONSIBLE bit.

In addition to the fault-tolerance measure noted above, KICK2 requires an acceptance test in order to ensure that control is indeed transferred to another triad in the event of retirement. The requirements for the acceptance test are shown in table 3.4, and the N-S diagram is shown in figure 3.4. As noted in table 3.4, the acceptance test is executed immediately after the completion of KICK. If the value in TRIAD.BUSY(RG4) is less than or equal to the value of the triad which has just executed KICK, then no other triad will respond to the IPC interrupt to take the R4 dispatcher. Under such circumstances, the R4 rate group will be restarted when the next R4.TICK.TIME is reached, and invocation of the alternate dispatcher is necessary.

Table 3.4. Requirements for the KICK Acceptance Test

1. The KICK acceptance test will be executed immediately after the KICK of the R4 dispatcher as a result of retirement occurs.
2. The acceptance test will read TRIAD.BUSY(RG4), and compare it to the value of TRIAD.ID of the calling triad. If the value of TRIAD.BUSY(RG4) is less or equal to TRIAD.ID, then the alternate dispatcher will occur.
3. This acceptance test will not be executed when KICK is used to transfer the R4 dispatcher to another triad under other circumstances.

3.4. R4.RESPONSIBLE ACCEPTANCE TEST

The purpose of this acceptance test is to ensure that the R4.RESPONSIBLE flag has been set in at least one triad before the conclusion of the execution of the R4 dispatcher. The N-S diagram and procedure requirements are given in figure 3.5 and table 3.5 respectively. The test first ensures that the value of TRIAD.COUNTER, the variable used to determine what triad will be responsible for starting the next frame, is within the expected range (0 to 2). If no other triads are executing the R4 dispatcher, this procedure checks that both R4.RESPONSIBLE is TRUE and that the difference between the current time (TIME.NOW) and the time to the beginning of the next frame (R4.TICK.TIME) is no more than the R4 frame length (R4.PERIOD).

If another processor triad is still executing the R4 dispatcher, then two or more triads would be R4.RESPONSIBLE, a condition which could result in a number of adverse consequences upon the restart of the R4 dispatcher. It is for this reason that this acceptance test will invoke the alternate dispatcher if TRIAD.COUNTER is not zero when R4.RESPONSIBLE is TRUE.

Table 3.5. Requirements for the R4.RESPONSIBLE Acceptance Test

1. The test is invoked by the R4 dispatcher immediately before the RESUME(0) statement.
2. The test will invoke the alternate dispatcher if any of the following conditions are met:
 - (a) TRIAD.COUNTER < 0 or TRIAD.COUNTER > 2
 - (b) TRIAD.COUNTER = 0 AND R4.RESPONSIBLE = FALSE
 - (c) R4.TIME.TICK - TIME.NOW > R4.PERIOD
 - (d) TRIAD.COUNTER >= 1 AND R4.RESPONSIBLE = TRUE

3.5. UNINTERRUPTIBLE CODE ACCEPTANCE TEST

Appendix B lists the series of assembly language routines which are called by the dispatcher or associated routines and which have sections of uninterruptible code. As is shown in table B-2, six of these routines have conditional branches or loops within them, and the possibility of an infinite loop due to a software logic, environment, or execution error exists. If the R4.RESPONSIBLE triad is involved, the R4 Interrupt will not be acted upon.

The remainder of the system can be in one of six configurations under this condition:

1. Two other triads idling.
2. Two other triads working on other applications routines.
3. One other triad working, one retired.
4. One other triad idling, one retired.
5. Both triads retired.
6. One triad working and one idle.

With the exception of 5, these configurations may be grouped into the following (non-mutually exclusive) categories:

- (I) At least one triad working.
- (II) At least one triad in the idle process

The two acceptance tests described below are to be run under the applicable conditions.

3.5.1. At least one triad working

If at least one other triad is performing a task, it will eventually be interrupted or will go to an idle state. If the triad receives a timer interrupt, control will pass to the timer interrupt handler. The acceptance test whose requirements are described in table 3.6 and depicted in the N-S diagram of figure 3.6 is appended at the end of the timer interrupt handler, and will evaluate whether the R4 dispatcher should be restarted. If not, this triad invokes the alternate dispatcher.

There are two conditions which will cause invocation of the alternate dispatcher under these conditions:

- (a) If there is still an R4.RESPONSIBLE triad (i.e. the R4 dispatcher has not been restarted) at the expected restart time plus an allowance DELTA, then the alternate dispatcher will be invoked;
- (b) If there are no triads running the R4 dispatcher beyond the allowed time, the alternate dispatcher will be invoked.

3.5.2 At least one triad in the idle process.

All triads in the idle mode will subtract the current time (TIME.NOW) from the time to the new R4 frame (R4.TICK.TIME). If the difference exceeds the expected time for the R4 dispatcher to invoke the KICK routine to start up other triads (denoted as DELTA.KICK) and other triads are available for running the R4 dispatcher, the idling triad will invoke the alternate dispatcher.

The idle mode acceptance test provides coverage for the following two situations (1) when the R4 dispatcher is stuck in an uninterruptible infinite loop after restarting the current frame or (2) when the last triad leaving the R4 dispatcher is stuck prior to designating itself as R4.RESPONSIBLE. The reader should note that the STUCK IN R4 acceptance test provides coverage in the event that the dispatcher is unduly delayed, but not under this condition.

TABLE 3.6. Requirements for the Uninterruptible Code Acceptance Test -
At Least One Triad Is Working

1. The Uninterruptible Code acceptance test will be invoked at the beginning of the TIMER.INTERRUPT.HANDLER routine.
2. A constant DELTA defined as expected time to enter the R4 dispatcher and set R4.RESPONSIBLE to FALSE will be defined during system initialization.
3. The test will read R4.TICK.TIME and TIME.NOW. If $TIME.NOW - (R4.TICK.TIME + DELTA)$ is less than or equal to zero, a normal return will be effected.
4. If $TIME.NOW - (R4.TICK.TIME + DELTA)$ is greater than zero and the triad is R4.RESPONSIBLE, the test will determine the contents of the APSD.
 - a. If the APSD = R4.PSD, a normal return will occur.
 - b. If the APSD is any other quantity, the alternate dispatcher will be invoked.
5. IF $TIME.NOW - (R4.TICK.TIME + DELTA)$ is greater than zero and the triad is not R4.RESPONSIBLE, the test will check the TRIAD.COUNTER (i.e. the number of triads executing the R4 dispatcher).
 - a. The triad will determine whether any other triad is R4.RESPONSIBLE. If no other triad is R4.RESPONSIBLE and the TRIAD.COUNTER is greater than or equal to 1, then a normal exit will occur.
 - b. If another triad is designated as R4.RESPONSIBLE beyond this time limit, the alternate dispatcher will be invoked.
 - c. If the TRIAD.COUNTER is zero, the alternate dispatcher will be invoked.

TABLE 3.7. Requirements for the Uninterruptible Code Acceptance Test -
At Least One Triad Is In the Idle Process

1. This test will be executed constantly when the triad is in the idle process.
2. The triad will determine the difference between TIME.NOW and R4.TICK.TIME.
 - a. If this difference is less than DELTA.KICK, the expected time to execute the KICK instruction to restart another triad, a normal return will occur.
 - b. If this difference is greater than DELTA.KICK the test will determine the number of triads available to execute the R4 dispatcher (by means of the TRIAD.STATUS(RG4) array) and the number executing the R4 dispatcher (by means of the TRIAD.COUNTER variable). If this difference is greater than zero, the test will invoke the alternate dispatcher.

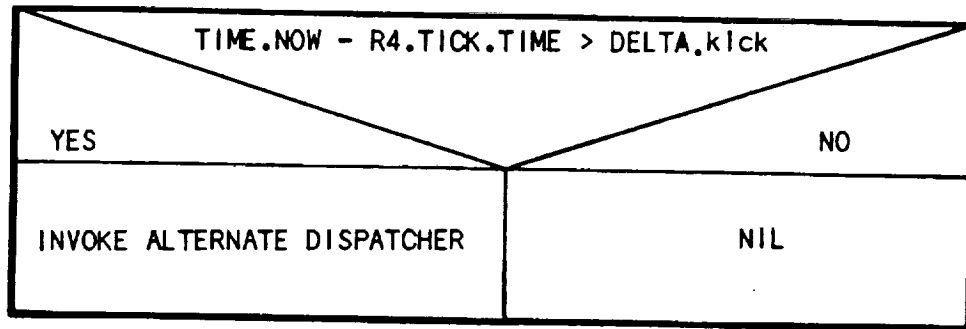


FIGURE 3.7. Uninterruptible Code Acceptance Test: Triad Idling

3.6. RETIREMENT ACCEPTANCE TEST

Prior to execution of its task list, the R4 dispatcher executes the configuration program which generates commands for retirement of faulty triads. The actual setting of control variables and the command to retire is performed by the R4 dispatcher itself, however, and this function is covered by the Retirement acceptance test.

Should a triad be commanded to retire, the dispatcher will zero appropriate bits in the TRIAD.BUSY and TRIAD.STATUS arrays, pass the responsibility for the R4 frame restart to another triad if the retired one is R4.RESPONSIBLE, and exit the R4 dispatcher. The Retirement acceptance test checks the TRIAD.CMND and TRIAD.STATUS words of all triads to ensure that a faulty triad has both successfully retired and properly executed its acceptance test.

The Retirement acceptance test is executed immediately prior to the RESUME(0) statement at the conclusion of the R4 dispatcher. If no retirement commands have been issued, the acceptance test is exited. However, if the configuration task issues a retirement command to the triad, the acceptance test checks the TRIAD.STATUS bits for the triad. If they are set to FALSE, the retirement directive has been successfully executed and the test will take a normal exit. However, if this bit is TRUE, the acceptance test will test for a previous failure. If no previous failures of this kind have occurred, the acceptance test will set a failure indicator and set the TRIAD.STATUS word. Any subsequent retirement failures would terminate execution of the primary dispatcher. Table 3.8 and figure 3.8 show the requirements and N-S diagram for the Retirement acceptance test.

TABLE 3.8. Requirements for the Retirement Acceptance Test

1. The Retirement acceptance test will be invoked immediately prior to the RESUME(0) statement concluding execution of the R4 dispatcher.
2. The acceptance test will read the TRIAD.CMND word from main memory. If the command does not indicate retirement, the test will exit.
3. If the TRIAD.CMND is GO.TO.IDLE for of any processor, the acceptance test will read the TRIAD.STATUS word.
4. If the TRIAD.STATUS(RG4) is FALSE, the acceptance test exits normally.
5. If TRIAD.STATUS(RG4) is TRUE, the acceptance test checks a failure indicator designated as RET.FAIL.
 - a. If RET.FAIL is FALSE, the acceptance test sets it to TRUE, and sets the TRIAD.STATUS to FALSE.
 - b. If RET.FAIL is TRUE, the test invokes the alternate dispatcher.
 - c. RET.FAIL is set to FALSE as part of system initialization.

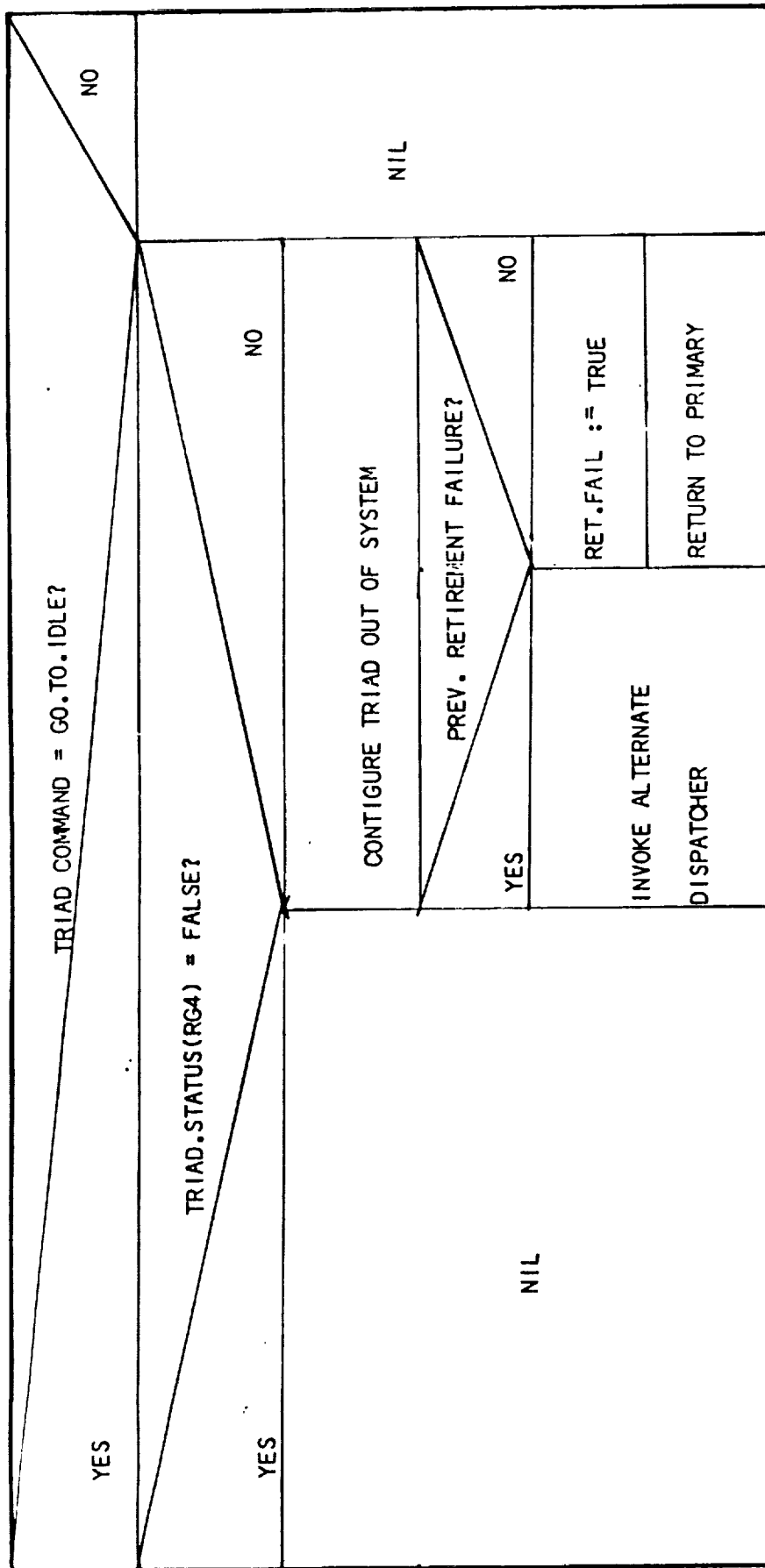


FIGURE 3.8. Retirement Acceptance Test

SECTION 4 - ALTERNATE DISPATCHER

This section covers the requirements and design of the alternate dispatcher as well as the conditions under which it returns control to the operating system. The dispatcher requirements are based on refs. 4 and 5. The design of the alternate dispatcher is intended (1) to be as simple as possible while fulfilling the necessary requirements and (2) to be independent of the primary dispatcher.

4.1. ALTERNATE DISPATCHER REQUIREMENTS

The alternate dispatcher performs the following functions

1. Initialization of system configuration variables.
2. Identification of lead, support, and inactive triads
3. Sequencing and Dispatching of I/O.
4. Dispatching of applications routines in a predetermined order.
5. Identifying conditions for return to the primary dispatcher.
6. Acceptance tests and aborts

Startup Initialization of the alternate dispatcher

Because the system will already be under "warm start" conditions at the time the alternate dispatcher is invoked, relatively little initialization will be performed by the alternate dispatcher. It is anticipated that other systems software, e.g. the configuration controller, reading and setting of error latches, the IPC interrupt routines, and supporting ASM routines, will not be affected by a dispatcher failure. Startup initialization includes the following:

- alternate dispatcher task list
- setting the alternate execution counter and repetition limit
- informing other triads of the invocation of the alternate dispatcher
- saving the state of the triads, busses, and memories

Identification of lead and support triads

The only assumption on the number of operational processors made in the design of the alternate dispatcher is that at least one triad will be functional. This triad will be responsible for calling the critical tasks of all rate groups in the appropriate sequence. Non-Critical tasks from all rate groups will run on a second triad if available. Other operational processors will be designated as

spares.

The triad invoking the alternate dispatcher will designate itself as lead triad, and will invoke the IPC interrupt routine to inform other triads of the dispatcher change. The first triad responding will be designated support triad; remaining processors will be spares. The lead triad will continue to run the alternate dispatcher; the support triad will run all non-critical applications routines in a fixed order.

Sequencing and dispatching of I/O

The total responsibility for the reading and writing of I/O buffers to the MIL-STD-1553 bus will rest with the first critical applications routine to be run on the R4 iteration group. This routine will have responsibility for designating even and odd buffer storage areas and performing the appropriate updating and sampling as required by the applications routines. In the absence of detailed specifications of the MIL-STD-1553 protocols and applications routines developed for the FTMP, no further detailed requirements will be specified.

Sequencing and Dispatching of applications routines

Because only a single triad will be used to dispatch the critical task list and a second triad will be used to implement an independent set of non-critical tasks, the following features which were a necessary part of the primary dispatcher were eliminated in the interest of simplicity and independence for the alternate dispatcher:

- (1) the recognition of constraints for applications routines
- (2) the starting of new frames at fixed time intervals
- (3) LOCK and UNLOCK functions for memory locations

The fact that features (1) and (3) are not implemented is inconsequential when only one triad invokes all applications routines. The loss of feature (2) implies that it is no longer correct to speak of a rate group or frames. Thus, in the alternate dispatcher, rate group is replaced with "iteration group" and frame is replaced with "iteration", i.e. R3 iteration group, iteration count, etc.

The alternate dispatcher will maintain the execution order of the primary routine by means of a task list which is placed into cache memory as part of the initialization process. The task list is divided into three iteration groups similar to the rate groups of the primary routine. The R4 iteration group, placed at the beginning of the list, is executed first. The R3 iteration group is executed every second iteration, and the R1 group is executed every eighth iteration.

Acceptance Test and Restarting of the Primary Dispatcher

At the conclusion of each iteration, the alternate dispatcher will run its acceptance test, and, if passed, will increment an execution counter and re-invoke the primary dispatcher if the execution limit has been reached. If

the alternate dispatcher fails, an alternate failure flag will be set, and control passed to another triad. If the routine fails a second time, then the processor will enter an abort routine.

4.2. DESCRIPTION OF THE ALTERNATE DISPATCHER

Figure 4.1 is an N-S diagram of the alternate dispatcher. When it is first invoked, this routine engages in the initialization and startup activities described above. It then enters the task execution loop in which the critical words are reset, the iteration count incremented, and the tasks from the various rate groups are run. At the conclusion of an iteration, the alternate dispatcher acceptance test (similar to that of the primary dispatcher) is run.

In order to achieve a further degree of independence, the alternate dispatcher calls the applications routines rather than "stringing" them by means of the PSD scheme used in the primary. Prior to the calling of applications routines, the interval timer is set using the START.R4.TIMER procedure (for all iteration groups). For the sake of simplicity and reliability, a single time will be used in this routine rather than reading a time limit from the task control block. If a timer interrupt occurs, control is passed back to the dispatcher which then calls the next task. In addition to not providing for the checking of constraints as noted above, the alternate dispatcher has no provisions for checking on the task return code or frame count. As was the case in the primary dispatcher, applications routines are expected to set a bit in the appropriate iteration group critical word and to contain internal recovery blocks.

The first failure to complete all critical tasks in an iteration will cause the dispatcher to be transferred to the support triad and the suspension of execution of non-critical tasks. If the failure persists, execution of the alternate dispatcher ceases and the system enters the abort routine.

Because of the limited capabilities of the alternate dispatcher, it is desirable to return to the primary routine as soon as possible. As a result, a repetition limit, based on the number of previous failures of the primary routine, is set during the startup initialization. When the dispatcher completes an iteration, it increments an alternate execution counter which is then compared to the repetition limit. Once the limit is reached, control is passed back to the primary procedure and the system restored to its original state (unless hardware failures have occurred in the intervening time). As was the case prior to invocation of the alternate dispatcher, acceptance tests will continue to monitor all aspects of the operation of the dispatcher and invoke the alternate once again upon detection of any failures.

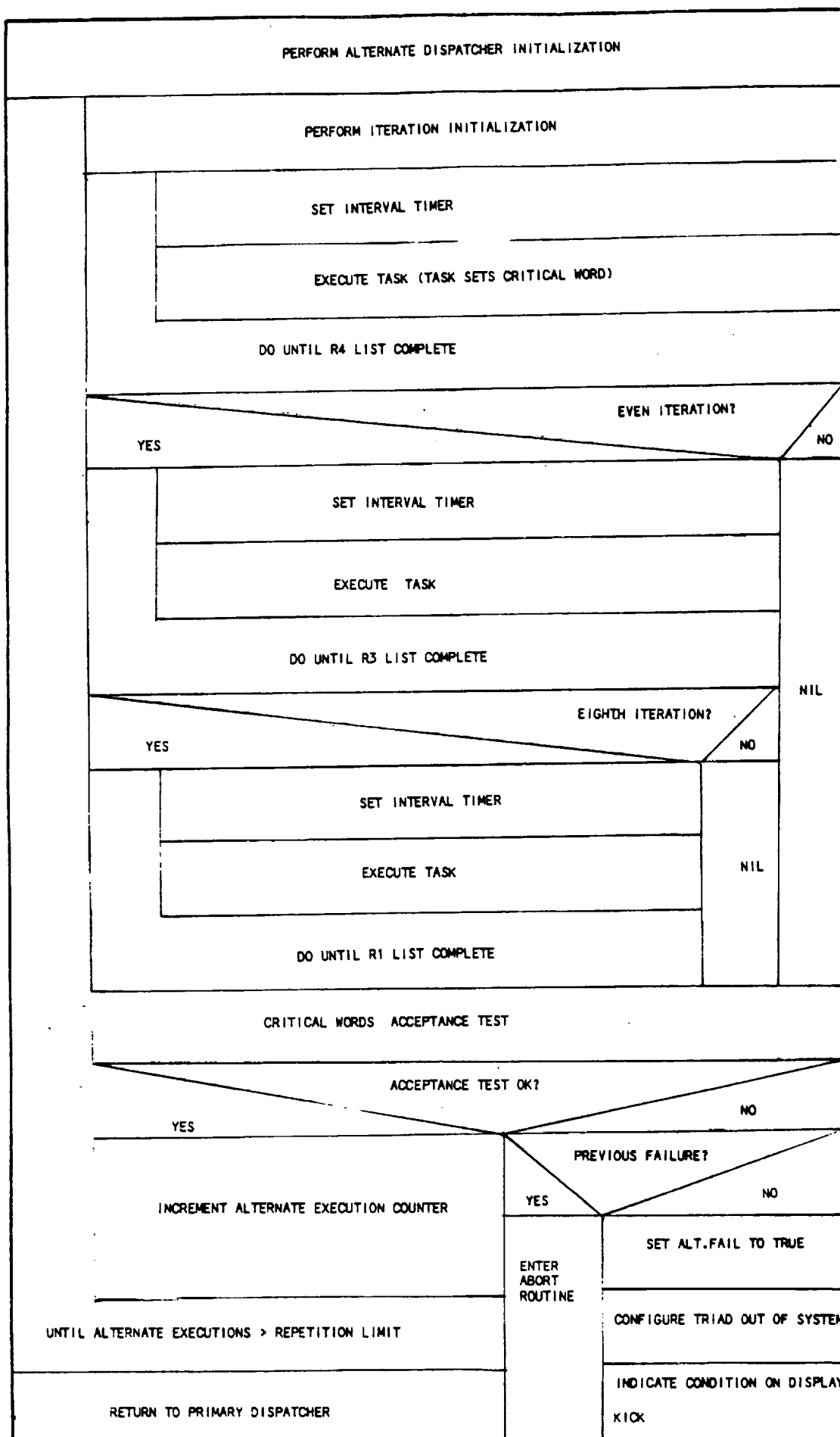


FIGURE 4.1. Alternate Dispatcher

REFERENCES

1. A. L. Hopkins, T. B. Smith, and J. H. Lala, FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft, Proceedings of the IEEE, Vol. 66, No. 10, pg. 1221, October, 1978
2. J. J. Horning, et. al., "A Program Structor for Error Detection and Recovery", Proceedings of the Conference on Operating System, Theoretical and Practical Applications, IRIA, pp. 174-193, April, 1974
3. FTMP Operating System Routines listing of 12 August, 1980
4. J. Lala, "Requirements for the FTMP Dispatcher", FTMP memo 9-79, CSDL, June 18, 1979
5. H. Hecht, "Introduction to the Use of Acceptance Tests for the FTMP Dispatcher", SoHaR memorandum, January, 1979
6. CSDL, FTMP Principles of Operation Volume II, Rev. 1.1, January, 1980
7. SoHaR, Inc., Engineering Report on Fault Tolerant Software for the FTMP, January, 1980

APPENDIX A - NEW VARIABLES REQUIRED

VAR. NAME	ROUTINE	PURPOSE
MINSLIP	SLIP ACCEP. TEST	maximum absolute value SLIP can have prior to the invocation of the alternate dispatcher
MAX.R4.TIME	INTERVAL TIMER ACCEP. TEST	Maximum value that can be loaded into an R4 timer (i.e. maximum execution time for an applications routine)
R1.CRIT.WD	DISPATCHER ACCEP. TEST	Indicator that R1 critical tasks have been dispatched
R3.CRIT.WD	DISPATCHER ACCEP. TEST	Indicator that R3 critical tasks have been dispatched.
R4.CRIT.WD	DISPATCHER ACCEP. TEST	Indicator that R4 critical tasks have been dispatched.
RX.INIT.CRIT	CRITICAL WORD RESET ACCEP. TEST	Initial values of critical words
OLD.FRAME, NEW.FRAME	FRAME COUNT ACCEP. TEST	Lead and lag counters to ensure that frame counter is appropriately incremented
FRAME.FAIL	FRAME COUNT ACCEP. TEST	Counter for number of errors in incrementing frame count
R4.APP.COUNT	STUCK IN R4 ACCEP. TEST	Counter of dispatched R4 tasks
ALT.EXEC.COUNT	ALTERNATE DISPATCHER	Counts executions of alternate dispatcher.
REP.LIMIT	ALTERNATE DISPATCHER	Repetition limit for alternate dispatcher after which primary is re-invoked.
PRIM.FAIL.COUNT	ALTERNATE DISPATCHER	Counts primary dispatcher failures.
PREV.ALT.FAIL	ALTERNATE DISPATCHER	Indicator of previous alternate dispatcher failure
TASK.LIST	ALTERNATE DISPATCHER	Array of task identifications for alternate dispatcher.

VAR. NAME	ROUTINE	PURPOSE
DELTA	UNINTERRUPT. CODE ACCEP. TEST	Time after new frame start for R4.RESP flag to be set to FALSE
DELTA.KICK	UNINTERRUPT. CODE ACCEP. TEST	Time after new frame start for idling tried to start the R4 dispatcher, if it is not R4 responsible

APPENDIX B - UNINTERRUPTIBLE ASM ROUTINES

Because failure detection by the proposed acceptance tests is contingent upon the implementation of the R4 interrupt, sections of code in which this interrupt is disabled are of significance. Failure to exit from these routines would result in inhibition of the R4 rate group restart, and a major but undetected failure would result if the "hung up" triad is R4 responsible.

ASM routines called by the AED rate group dispatchers and associated procedures were checked for the presence of a SWAPMSK command that would disable all (and hence R4) interrupts. The following procedures were examined:

- DISP.R4
- DISP.R3.R1
- SELECT.TASK
- EXECUTE
- HOLD.TIMER
- RELEASE.TIMER
- READ.EL
- SET.BIT
- KICK
- LOCK
- UNLOCK

Table B.1 shows the list of ASM routines in these procedures, and an indication of whether all interrupts are disabled. Table B.2 shows the eight ASM routines which disabled interrupts, their function, and notes on the complexity of the code during the interrupt defeat.

Software failures in routines with straight-line code or with a limited number of unconditional jumps are unlikely. However, when loops or conditional jumps are present, the possibilities of failure increase. Based on Table B.2, it is possible to rule out HOG.BUS and possibly RELEASE.BUS as sources of concern. However, for the remainder of the ASM routines, it is necessary to determine whether the R4 interrupt should be enabled, whether other means exist to detect the R4 interrupt, or whether additional measures (which will increase the complexity of the operating system) are appropriate.

Table B.1

AED Procedures and Associated ASM Routines

AED Procedure	ASM Routine	Uninterruptable Code
DISP4	READ	YES
	WRITE	YES
	PEND	YES
	RESUME	YES
	SWAPMASK	NO
SELECT.TASK	READ	YES
	WRITE	YES
	SWAPMASK	NO
EXECUTE	READ	YES
	WRITE	YES
	ACTIVATE	YES
	ASNBIT	NO
HOLD.TIMER	SWAPMASK	NO
RELEASE TIMER	SWAPMASK	NO
	WRITE	YES
TIMER.INT.HANDLER	PEND	YES
	RESUME	YES
READ.EL	READ	YES
	HWRITE	YES
	WRITE	YES
SET.BIT	HOG.BUS	YES
	ASNBIT	NO
	WRITE	YES
	RELEASE.BUS	YES
CLEAR.T.EL	SREAD	YES
KICK	HOG.BUS	YES
	READ	YES
	HWRITE	YES
	RELEASE.BUS	YES
LOCK	HOG.BUS	YES
	READ	YES
	WRITE	YES
	RELEASE.BUS	YES
	SWAPMASK	NO

Table B.1 (continued)

AED Procedures and Associated ASM Routines

AED Procedure	ASM Routine	Uninterruptable Code
UNLOCK	SWAPMASK	NO
	HOG.BUS	YES
	READ	YES
	ASNBIT	NO
	WRITE	YES
	HWRITE	YES
	RELEASE.BUS	YES



Table B.2

ASM Routines Containing Uninterruptible Sections of Code

ASM Routine	Called By	Description	Note
READ	DISPATCHERS SELECT.TASK EXECUTE READ.EL KICK LOCK UNLOCK	Reads from main memory; Interrupt inhibited while data is transferred.	1
WRITE	DISPATCHERS SELECT.TASK EXECUTE RELEASE.TIMER SET.BIT LOCK UNLOCK	Writes to main memory; Interrupt inhibited while data is transferred	1
ACTIVATE	EXECUTE	Store new PSD, mask string if necessary	2
PEND	TIMER.INT.HANDLER	String new PSD behind current PSD	3
RESUME	DISPATCHERS TIMER.INT.HANDLER	Resume previously Interrupted task (take old PSD and make current).	4
HWRITE	READ.EL	Write to hardware register, inhibits interrupt during hardware transfer.	5
HOG.BUS	SET.BIT KICK LOCK	Increment HOG.WORD	6
RELEASE.BUS	SET.BIT KICK LOCK	Decrement HOG.WORD	7

Notes for Table B.2

1. SWPMSK after BEGIN label (line 58) inhibits interrupt. Several jumps in each transfer.
2. Part of KERNEL. 4 jumps (2 conditional) in uninterruptible sequence.
3. 1 loop, 1 conditional jump, 17 instructions in uninterruptible sequence.
4. Part of KERNEL (not yet clear how entered). SWPMSK assumed prior to execution; 15 statements, 2 jumps prior to SWPMSK which would re-enable interrupts.
5. Numerous conditional jumps (depending on amount of data to be transferred) and ASM statements in uninterruptible execution sequence.
6. 7 statements, no jumps or loops, in uninterruptible sequence.
8. 1 conditional jump, 1 unconditional jump, 22 statements in uninterruptible sequence.

1. Report No. NASA CR-166070		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle Fault-Tolerant Software for the FTMP				5. Report Date March 1984	
				6. Performing Organization Code	
7. Author(s) Herbert Hecht and Myron Hecht				8. Performing Organization Report No.	
9. Performing Organization Name and Address The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge, MA 02139				10. Work Unit No.	
				11. Contract or Grant No. NAS1-15336	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546				13. Type of Report and Period Covered Contractor Report	
				14. Sponsoring Agency Code 505-34-13-33	
15. Supplementary Notes Langley Technical Monitor: Charles Meissner, Jr. Prepared by SoHar Incorporated under Subcontract 564 to Charles Stark Draper Lab.					
16. Abstract <p>The work reported on here provides protection against software failures in the task dispatcher of the FTMP, a particularly critical portion of the system software. Faults in other system modules and application programs can be handled by similar techniques but are not covered in this effort. Goals of the work reported on here are: (1) to develop provisions in the software design that will detect and mitigate software failures in the dispatcher portion of the FTMP Executive and, (2) to propose the implementation of specific software reliability measures in other parts of the system.</p> <p>Beyond the specific support to the FTMP project, the work reported on here represents a considerable advance in the practical application of the recovery block methodology for fault tolerant software design.</p>					
17. Key Words (Suggested by Author(s)) Fault Tolerant Software, Recovery Block, Fault Tolerant Multiprocessor, Fault Tolerant Dispatcher				18. Distribution Statement  	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 82	22. Price		